UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FIN DE GRADO

Diseño e implementación de un laboratorio Big Data

Autor:
JESÚS MANUEL GALLEGO ROMERO

Supervisor:
DR. DANIEL DÍAZ SÁNCHEZ

Grado en Ingeniería en Tecnologías de Telecomunicación

Departamento de Ingeniería Telemática

25 de septiembre de 2016

«Excepto la nobleza de sangre, todo lo demás puede adquirirse por medio del esfuerzo: el genio, la sabiduría, la inteligencia.»

Fiódor Dostoyevsky

UNIVERSIDAD CARLOS III DE MADRID

Resumen

Escuela Politécnica Superior Leganés
Departamento de Ingeniería Telemática

Grado en Ingeniería en Tecnologías de Telecomunicación

Diseño e implementación de un laboratorio Big Data

by Jesús Manuel Gallego Romero

El incremento del uso de las nuevas tecnologías y la forma en la que estas tecnologías se usan presentan un nuevo escenario dentro del mundo de las TIC debido a la cantidad y variedad de datos generados. Para inferir información útil a partir de todos estos datos entra en escena el análisis de Big Data, cuyo objetivo se basa en resolver el problema mediante el uso de técnicas y procedimientos diferentes a los tradicionales. Este trabajo se centra en la creación de un entorno donde estas prácticas puedan llevarse a cabo de manera controlada para ofrecer la posibilidad de la creación de aplicaciones de Big Data y la posterior comprobación del comportamiento de estas aplicaciones según diferentes configuraciones. Se discutirá en el proyecto cuáles son las diferentes opciones a la hora de implementar un cluster de este tipo y se propondrá un diseño con una de esas tecnologías que posteriormente se llevará a la práctica mediante la instalación en un laboratorio real de la Universidad Carlos III de Madrid.

The increase of use of new technologies and the way these technologies are used introduces a new scenario in the field of computing, due to the quantity and variety of generated data. In order to extract valuable information from all this data Big Data analysis comes on the scene. Big Data analysis aims to solve this problem by using new techniques different from the traditional ones. This projects is based in the implementation of a scenario where these techniques can be deployed and tested in a controlled way in order to give the opportunity to developers to create Big Data apps and also test the way this applications behave under certain configurations. In the project several options will be presented at the moment of creating a Big Data cluster and also a design will be proposed for the implementation with one of these technologies. This design will also be taken to practice by implementing it in a real laboratory of the University Carlos III of Madrid.

Acknowledgements

En primer lugar, me gustaría agradecer a Dani, mi tutor. Gracias por guiarme en este proceso, por enseñarme tantas cosas, tanto del proyecto como ajenas a él.

Quería agradecer también a los técnicos del laboratorio de telemática el trabajo dedicado.

Gracias en especial a mis amigos y amigas, que durante este tiempo seguro os habré molestado con cosas del proyecto y siempre habéis estado ahí para ayudarme.

Gracias a mis compañeros de clase, Nacho, Víctor, Irene y Elena. Por estos cuatro años juntos que espero que sean muchos más. Espero también haberos aportado al menos la mitad de la mitad de lo que vosotros me habéis aportado a mí: muchas gracias chicos.

Gracias a mi compañero de todo durante estos años, Santi. Por todo lo que hemos peleado juntos, y mucho más por todo lo que hemos disfrutado juntos. Llegó el momento de abrir el vino.

Gracias a ti, Alba. Por aguantar todas mis quejas, por escucharme y tener siempre unas palabras para animarme.

Gracias a mi hermana, Elena por, aunque tú no lo sepas, motivarme día tras día.

Gracias a mis padres, Antonio y Mariángeles. Sin vosotros nada hubiera sido posible. Muchas gracias por darme esta oportunidad. Espero algún día ser capaz de recompensaros.

Por último, me gustaría dedicar este trabajo a mi abuela. Porque sé que siempre has confiado en mí. Estoy seguro que sin esa ayuda que tú y yo sabemos no habría podido llegar hasta aquí.

Gracias, para terminar, a todo aquél que haya contribuido a la realización de este proyecto.

Índice general

Índice de figuras

Índice de cuadros

List of Abbreviations

HDFS	**H**adoop **D**istributed **F**ile **S**ystem
YARN	**Y**et **A**nother **R**esource **N**egotiator
NDFS	**N**utanix **D**istributed **F**ile **S**ystem
RDBMS	**R**elational **D**atabase **M**anagement **S**ystem
FQDN	**F**ully **Q**ualified **D**omain **N**ame
QoS	**Q**uality of **S**ervice
JMX	**J**ava **M**anagement e**X**tensions
AMS	**A**mbari **M**etrics e**S**ervice
RPC	**R**emote **P**rocedure **C**all
ACL	**A**ccess **C**ontrol **L**ist
AM	**A**pplication **M**anager
NM	**N**ode **M**anager
LDAP	**L**ightweight **D**irectory **A**ccess **P**rotocol
API	**A**pplication **P**rogramming **I**nterface
GFS	**G**oogle **F**ile **S**ystem
SQL	**S**tructured **Q**uery **L**anguage
RPC	**R**emote **P**rocedure **C**all
TIC	**T**ecnologías de la **I**nformación y la **C**omunicación

Capítulo 1

Introduction and goals

The project is presented to the reader in this chapter. It introduces also the main objectives and motivations that have made this project possible. Finally, the structure of this paper is presented to make as easy as possible the reading.

1.1. Introduction

Technology and TIC have become in the last few years an essential part of society. In order to illustrate this increase in the usage of technology, several examples are presented:

- Thirty billions of content are shared in Facebook monthly, independently of their sizes.
- Currently, all the existing music could be stored in a hard drive worth 500 euros.
- During the last two years, the 90 % of the amount of the total stored data have been created.

This growth in the usage of technology and network connections represent at the same time an increase in the amount of data the network uses. The challenge for this data is not only to store it but also to be able to extract useful information from it, and Big Data analysis is based on that.

In order to store and analyze this huge amount of data, specific tools designed for that purpose are needed; and also professionals qualified to use them and avoid the creation of the known data tombs. This project is born from this idea, trying to approach to the Big Data analysis tools within a scenario easy to deploy. Behind this idea there are educative goals.

These tools used for the analysis work over set of machines, taking advantage in a collective way from the resources that each machine can give individually. Unfortunately, having access to a cluster is not easy and that is the reason why it is important to have a scenario where a cluster can be created and controlled.

One of the main intentions that have been wanted is to see if it compensates to face the difficulties that handling a cluster has over the possibilities that it offers.

Finally, it is interesting to be able to study the typical Big Data analysis tools even though if there is not possible to have access to a real cluster, this is having access to a virtualized scenario and being able to understand the possibilities it offers.

1.2. Socioeconomic environment

According to the **National Statistics Institute**, dated on October the second 2014, 74,4 % of homes had internet connection. For the first time in Spain this connection were not exclusively done from computers, but internet access from mobiles have overtaken the access from computers, being the data 77,1 % over 73,3 %, respectively.

But not only the access to the internet has increased, but also the way this connection is used. The 51,1 % of the population is active in social networks, and this means that half of the population not only consume content from the network but also generate it. This data proves the social change that technology is making, and the other way around.

Business management has also been shaken up. Those strategies used by the companies for their growth are known as business intelligence, and it is very related to Big Data during the last years. To put this into words, most inversions made in the stock exchange are done by decision algorithms running on top of Big Data clusters. This transactions are called High Frequency trading and represents the 50 % of the commercial volume of the stock exchange in New York.

1.3. Goals and motivation

The main objective pursued in this project is the implementation of a cluster equipped with the most popular tools of Big Data analysis. It is pretended this implementation to be an easy and guided process for anyone who wants to follow it and learn new concepts of this field.

This goal has a great relevance in the educative field, to bring the opportunity to a group of students of experimenting by their own the use of professional tools; being monitored and controlled by a teacher.

Besides this goal, it is pretended also to provide a scenario for developers to test their applications and check their behaviour. In order to measure the behaviour of these applications, the cluster should have different metrics which tell what is really happening in the computers. The possibilities offered would go from checking the effect of different programming styles, through the different configurations that can be applied to clusters, to the difference between executing modes, for example local and distributed.

Additionally, and completing the main goal of the project, which is the implementation of the cluster, it is wanted to make this implementation in the realest possible scenario. This would be formed by a real laboratory

with several machines and even different network segments. This objective would give a general view of the performance of a real cluster and would complete the educative motivation of the project. To fulfill this objective, it is required the implementation in the laboratory of telematics engineering of the university Carlos III of Madrid.

1.4. Memory content

This document is formed by 9 chapters and three appendices. The content of the document is going to be briefly described to ease the reading and the understanding of the project.

1.4.1. Chapter 1: Introduction and goals

This chapter is used as introduction and it presents the motivation and the main objectives of the project.

1.4.2. Chapter 2: State of the Art

In this chapter a general view of the state of Big Data is presented. Also the possibilities that gives Big Data analysis and some open source tools that can be used for that purpose.

1.4.3. Chapter 3: Design

This chapter details the original idea that was thought when the project started. It shows the design of a cluster with the necessary components to be totally functional in a real scenario, as it could be a company.

1.4.4. Chapter 4: Prototype

This chapter covers the description of the created cluster showing details of the installed components and the process of installation.

1.4.5. Chapter 5: Testing

This chapter details the tests applied to the created cluster and shows the cluster's response to different tasks.

1.4.6. Chapter 6: Project's history

In this chapter the different problems that appeared during the project are explained.

1.4.7. Chapter 7: Planning, budgeting and legal framework

In this chapter the organization followed during the project is detailed as well as the list of the used resources. With this list of resources there is also the budget of the project and the legal framework related to it.

1.4.8. Chapter 8: Conclusions and future work

This chapter compares the objectives proposed at the beginning of the project with the results obtained extracting some conclusions of the work done. Besides, it includes guides and recommendations towards where the project could be taken in further investigation.

1.4.9. Chapter 9: English summary

This chapter is a summary of the main parts of the project, which are the introduction, the state of art, the design, the prototype, the history of the project and the conclusion.

1.4.10. Appendix A: WordCount code

In this appendix it is included the code used to test the cluster.

1.4.11. Appendix B:Introducción

In this appendix the introduction of the project is presented in spanish.

1.4.12. Appendix C: Conclusions and future work

In this appendix the conclusions and future work are described in spanish.

Capítulo 2

Estado del Arte

2.1. Definición de Big Data

La primera pregunta que puede ser planteada es: ¿qué es Big Data? De acuerdo con IBM [4], Big Data se define como toda la información, estructurada o desestructurada, que no puede ser procesada o analizada mediante herramientas tradicionales.

Las principales características que definen Big Data son: *Volumen*, *Variedad* y *Velocidad*; conocidas como V^3. Estas características implican la necesidad de nuevas capacidades para procesar de manera eficiente esta nueva información.

En relación al *Volumen*, las previsiones de datos generados para el 2020 es de 35 zetabytes (un zetabyte equivale a 10^{21} bytes). Es un hecho que hoy en día existen más datos que nunca. Para hacerse una idea, Twitter genera 7 terabytes (TB) de datos diariamente mientras que Facebook hace lo mismo con 10 TB [5]; se puede afirmar que las cosas han cambiado en cuanto a Volumen.

La *Variedad* del Big Data representa todos los posibles tipos de datos. Los diferentes orígenes de estos, tales como sensores, datos de páginas web, archivos de log o redes sociales presentan un nuevo reto a la hora de analizarlos. Se estima que sobre el 80 % de los datos son desestructurados o semiestructurados. Twitter puede servirnos de ejemplo, pues estructura sus datos en forma de JSON y sin embargo el texto plano carece de estructura. Por otra parte, imágenes y vídeos no se pueden guardar de manera eficiente en bases de datos relacionales. Esta Variedad supone un problema para ambas tareas, tanto guardar como analizar los datos.

Trabajar con Big Data y su Volumen y Variedad introduce un nuevo paradigma: el procesamiento de flujo de datos o stream computing. Esto quiere decir, el procesamiento de Big Data no se aplica sobre datos estáticos; sino sobre datos que se encuentran en movimiento. Dependiendo de esos flujos de datos, se pueden diferenciar diferentes tipos de procesamiento, que son: procesamiento por lotes o batch processing, procesamiento cercano al tiempo real o near real-time processing y procesamiento en tiempo real o real real-time processing. Todo esto se relaciona con la característica de la *Velocidad*.

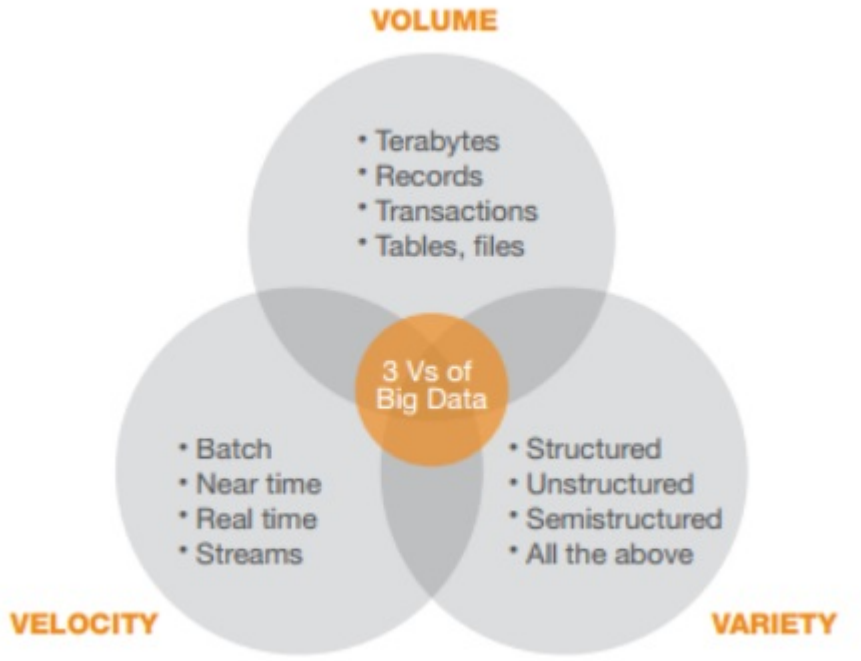


FIGURA 2.1: Las tres V's del Big Data: Volumen, Variedad y Velocidad.

2.2. Frameworks de Big Data

Con el fin de analizar Big Data, existen diferentes frameworks, contando cada uno con diferentes posibilidades. Entre aquellos de código libre, podemos encontrar Apache Hadoop, Apache Spark, Apache Flink, Apache Storm o Apache Samza [6]; entre otros. Apache Hadoop y Apache Spark son principalmente para procesamiento de datos por lotes (batch processing), aunque mediante Apache Spark se puede conseguir una emulación de procesamiento en tiempo real haciendo los intervalos de procesamiento muy pequeños. Apache Flink, Apache Storm y Apache Samza están orientados al procesamiento en tiempo real.

2.2.1. Apache Hadoop

Apache Hadoop ocupa la posición dominante en cuanto a procesamiento de Big Data. Se trata de un framework que permite el procesamiento distribuido de grandes sets de datos a través de un cluster de máquinas usando modelos de programación simples. Ha sido diseñado para escalar desde un sólo servidor a miles de máquinas, donde cada servidor aporta tanto poder de cálculo como almacenamiento local. En vez de estar diseñado para ejecutarse sobre hardware de alta fidelidad, la librería esta diseñada para detectar y manejar fallos en el nivel de aplicación, ofreciendo de esta forma un servicio de alta fidelidad sobre un cluster de máquinas en el cual cada una de estas máquinas puede ser vulnerable a fallos.

El proyecto Apache Hadoop deriva del proyecto MapReduce de Google y del sistema de ficheros de google GFS (Google File System). Incluye diferentes módulos [7]:

- ***Hadoop Common***: Esta utilidad permite el soporte de otros módulos de Hadoop. Estas librerías dan una abstracción a nivel de SO y sistema de ficheros; además contiene los ficheros de Java y scripts necesarios para arrancar Hadoop.
- ***Sistema Distribuido de Ficheros de Hadoop (HDFS)***: Llamado así por su nombre en inglés: Hadoop Distributed File System. Sistema de ficheros distribuido que proporciona un alto throughput para el acceso a datos de la aplicación y un alto ancho de banda agregado a través del cluster.
- ***Hadoop YARN***: Framework que permite establecer horarios, prioridades y gestión de recursos del cluster.
- ***Hadoop MapReduce***: Sistema basado en YARN usado para el procesamiento de grandes sets de datos usando el paradigma de programación clave-valor.

Se pueden encontrar otros proyectos de Apache relacionados con Hadoop y que se integran de manera sencilla con este framework:
a) *Ambari* **b)** *Avro* **c)** *Cassandra* **d)** *Chukwa* **e)** *Hbase* **f)** *Hive* **g)** *Mahout* **h)** *Pig* **i)** *Spark* **j)** *Zookeeper*

2.2.2. Apache Spark

Apache Spark es un sistema rápido de computo distribuido de propósito general. Proporciona APIs a alto nivel en Java, Python y R, y un motor optimizado que soporta la ejecución de grafos. Soporta también un gran conjunto de herramientas de alto nivel como pueden ser Spark SQL para procesamiento de datos estructurados, MLlib para aprendizaje máquina, GraphX para procesamiento de grafos y Spark Streaming para procesado de flujos de datos [8].

Las ventajas más significativas para justificar el uso de Apache Spark son las siguientes:

- ***Velocidad***: Spark ha sido diseñado para el rendimiento, pudiendo ser 100x más rápido que Hadoop en el procesamiento de grandes sets de datos mediante el uso de cálculo en memoria y otras optimizaciones. Spark también es rápido con los datos guardados en disco, y actualmente posee el récord mundial al ordenar datos en disco a gran escala.
- ***Facilidad de uso***: Spark dispone de APIs de fácil uso para operar con grandes sets de datos, como por ejemplo MLlib.
- ***Un motor unificado***: Spark viene empaquetado con librerías de alto nivel, incluyendo soporte para peticiones SQL, flujo continuo de datos (streaming), aprendizaje máquina y procesado de grafos. Estas librerías incrementan la productividad del desarrollador y pueden ser combinadas para crear flujos de trabajo complejos.

2.2.3. Apache Flink

Apache Flink es una plataforma de código libre para procesado distribuido tanto de flujo de datos como de intervalos (batch processing) [9]. El núcleo de Flink proporciona distribución de los datos, comunicación y tolerancia a fallos para cálculos distribuidos sobre flujos de datos.

Flink incluye varias APIs para el desarrollo de aplicaciones que usen el motor de Flink, que son:

- ***DataStream API*** para flujos continuos de datos en Java y Scala.
- ***DataSet API*** para el procesado de datos estáticos en Java, Scala y Python.
- ***Table API*** con expresiones de lenguaje basado en SQL para Java y Scala.

Flink agrupa también librerías para usos específicos:

- ***CEP***: librería para el procesado de eventos complejos.
- ***Machine Learning Library***: librería para aprendizaje máquina.
- ***Gelly***: API y librería para el procesado de grafos.

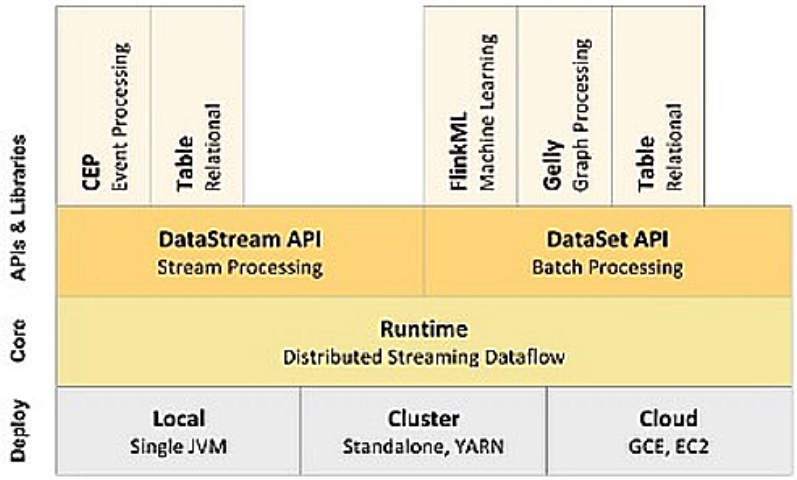


FIGURA 2.2: Ecosistema y estructura de Apache Flink.

2.2.4. Apache Storm

Apache Storm es un software de licencia libre orientado al cómputo distribuido en tiempo real [10]. Storm facilita el trabajo con flujos continuos de datos mediante el uso de procesamiento en tiempo real versus Hadoop, que realiza procesamiento en intervalos. Por otra parte, Storm puede ser usado con cualquier lenguaje de programación.

Storm tiene varios escenarios de uso: analíticas en tiempo real, aprendizaje máquina online, computación continua, RPC distribuido, entre otros. Los benchmarks han fijado la velocidad de Storm en tuplas de más de un millón de elementos procesadas por segundo por nodo. Storm es capaz de escalar, tolerar fallos, garantizar que todos los datos serán procesados y también presenta facilidad tanto para establecer el entorno como para operar con él.

Una topología de Storm consume flujos de datos y los procesa de forma compleja y arbitraria, repartiendo los flujos entre los estados del cálculo de la manera requerida.

2.2.5. Apache Samza

Apache Samza es un framework orientado al procesamiento distribuido de flujos de datos. Hace uso de Apache Kafka para mensajería y de Apache Hadoop YARN para la tolerancia a fallos, aislamiento de procesos, seguridad y gestión de recursos [11].

Las principales características de Apache Samza son:

- ***API Simple***: A diferencia de las APIs a bajo nivel de mensajería, Samza proporciona una API muy sencilla comparada con MapReduce llamada "process message".
- ***Estados Controlados***: Samza guarda y permite restablecer los estados de los procesadores de flujo. Samza ha sido concebido para poder guardar una gran cantidad de estados (varios gygabytes por partición).
- ***Tolerancia a fallos***: En el momento en que una máquina del cluster falla, Samza trabaja con YARN para migrar de manera transparente al usuario las tareas a otro servidor.
- ***Fiabilidad***: Samza utiliza Kafka para garantizar que los mensajes mantienen el orden en el que fueron escritos al ser procesados, y evitar que ningún mensaje se pierda.
- ***Escalabilidad***: Todos los niveles de Samza están particionados y distribuidos. Kafka proporciona flujos de datos ordenados, particionados y tolerantes a fallos. Los contenedores de Samza son ejecutados en el entorno creado por YARN.
- ***Pluggable***: A pesar de que Samza funciona con Kafka y YARN, Samza ofrece una API que permite ejecutar Samza con otros sistemas de mensajería y entornos de ejecución.
- ***Aislamiento***: Al trabajar con Apache YARN, Samza se beneficia del sistema de seguridad de Hadoop y de su aislamiento de recursos a través de los grupos de control de linux (Linux CGroups).

2.3. Ecosistema Apache Hadoop

A pesar de que Hadoop es conocido por MapReduce y su sistema de ficheros distribuido (HDFS, que fue renombrado de NDFS), el término Hadoop es usado también para una familia de proyectos relacionados que se encuentran bajo la infraestructura de la computación distribuida y el procesamiento de datos a gran escala [12].

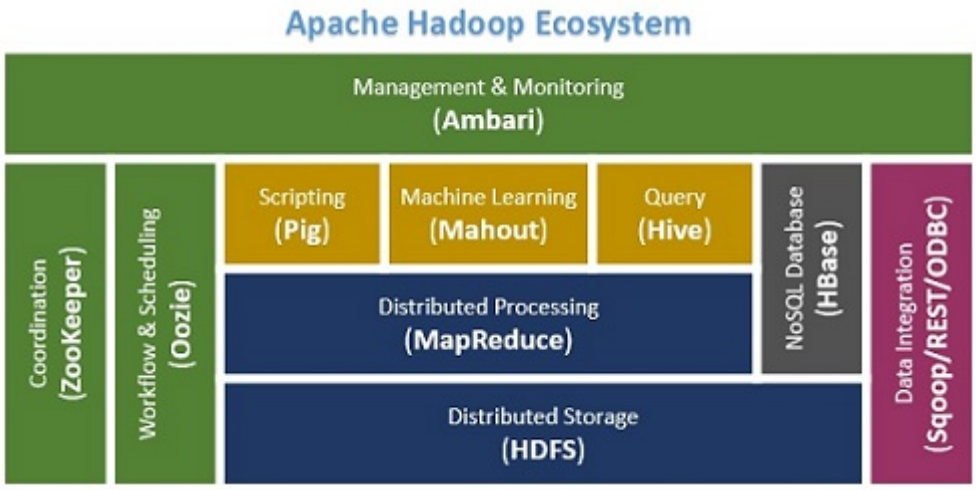


FIGURA 2.3: Diferentes componentes del Ecosistema Hadoop.

A continuación se presentan algunos componentes del ecosistema Hadoop. A pesar de que no constituyen la totalidad de dicho ecosistema, se trata de algunos de los componentes que se estudiarán en este trabajo: MapReduce, HDFS, YARN, Pig, Hive, HBase, ZooKeeper, Spark y Ambari.

2.3.1. MapReduce

MapReduce es un modelo de programación para el procesamiento de datos. Se trata de un modelo simple pero capaz de expresar y resolver problemas útiles. Los programas de MapReduce pueden estar escritos en diferentes lenguajes y están orientados a ser paralelizables [13].

MapReduce resulta útil para el procesamiento por lotes de terabytes o petabytes de datos almacenados en Apache Hadoop (HDFS). Los beneficios más destacables son:

- ***Simplicidad:*** Los desarrolladores pueden escribir las aplicaciones en Java, C++ o Python. Además, las tareas de MapReduce son fáciles de ejecutar.
- ***Escalabilidad:*** MapReduce es capaz de procesar petabytes de datos almacenados en HDFS en un clúster.
- ***Velocidad:*** MapReduce está orientado a la paralelización de las tareas.

- ***Recuperación:*** Se encarga de los posibles fallos. Cuando una copia no está disponible, una copia de esa tupla clave valor en otra máquina puede ser usada en la tarea. El encargado de esto es el JobTracker.
- ***Mínimo movimiento de datos:*** MapReduce mueve las tareas a los diferentes nodos del HDFS donde los datos están guardados y no al revés. De este modo, la ejecución se realiza en el nodo que posee los datos y se evita la congestión en el backbone del cluster debido al tráfico.

2.3.2. HDFS

HDFS es un sistema de ficheros diseñado para el almacenamiento de grandes archivos con modelos de acceso de flujo, se ejecuta en clusters y no exige hardware de alta fidelidad. En clusters actuales, se han llegado a almacenar archivos de petabytes.

La idea detrás de HDFS es el paradigma *escribe una vez, lee muchas veces*, que es considerado uno de los modelos más eficientes para el procesado de datos. Como ha sido mencionado anteriormente, no requiere de hardware de alta fidelidad debido a la tolerancia de fallos. Al ser diseñado para alcanzar un elevado throughput, esto puede significar una penalización en cuanto a latencia. La memoria total a la que el cluster puede aspirar está restringida por el namenode, debido a los metadatos que este debe guardar e indexar de todos los archivos del cluster.

2.3.3. YARN

YARN forma parte del núcleo de Hadoop permitiendo que varios motores de procesamiento como interactive SQL, procesamiento en tiempo real o por intervalos, trabajen sobre datos almacenados sobre una plataforma única, por ejemplo, HDFS o HBase [14].

YARN extiende el potencial de Hadoop a las nuevas tecnologías que surgen para el análisis de datos para que puedan tomar ventaja del procesamiento basado en coste que YARN realiza. Proporciona un framework consistente a desarrolladores con la finalidad de implementar aplicaciones que se ejecuten en Hadoop.

Al tratarse de un servicio central, YARN mejora la capacidad de cómputo de un cluster Hadoop en los siguientes aspectos:

- ***Acceso múltiple:*** YARN permite el uso de Hadoop a diferentes motores de procesamiento y acceso simultáneo al mismo set de datos.
- ***Utilización del cluster:*** Ofrece ventaja sobre el uso estático de recursos utilizado por las primeras versiones de MapReduce al contar con una administración dinámica de los recursos del cluster.
- ***Escalabilidad:*** YARN se centra en el proceso de organizar y establecer horarios para las tareas, dejando que el cluster escale hasta miles de nodos o petabytes de datos.

- ***Compatibilidad:*** YARN cuenta con compatibilidad hacia atrás para las tareas de MapReduce orientadas a Hadoop 1 con los actuales clusters y servicios que corren actualmente.

2.3.4. Pig

Apache Pig es una plataforma para analizar grandes sets de datos. Está formado por un lenguaje de alto nivel para expresar programas de análisis de datos, acoplado con una infraestructura para la evaluación de dichos programas. La propiedad más remarcable de Pig es que los programas están orientados a la paralelización, que permite el manejo de grandes sets de datos de manera eficiente [15].

Se puede dividir la estructura de Pig en dos:

- El lenguaje utilizado para expresar los flujos de datos, Pig Latin.
- El entorno de ejecución en el que corren los programas escritos en Pig Latin. Existen actualmente dos entornos: ejecución local en una única máquina virtual de Java y ejecución en un cluster Hadoop.

2.3.5. Hive

Apache Hive es una plataforma para ejecutar peticiones SQL sobre grandes volúmenes de datos guardados en HDFS [16]. Actualmente, empresas del tamaño de Facebook guardan sus datos de manera distribuida en HDFS y utilizan Hive para interactuar con estos datos. Si bien es cierto que SQL puede no ser ideal para todo tipo de problemas de Big Data, es una estructura de datos bien conocida y es la *lingua franca* en herramientas de bussiness inteligence.

Se pueden presentar las ventajas de Hive implementado sobre HDFS de Hadoop como las siguientes:

- ***Familiaridad:*** Al realizarse las peticiones con un lenguaje basado en SQL.
- ***Rapidez***.
- ***Escalabilidad:*** Conforme el volumen y la variedad de los datos aumenta, se puede aumentar el número de máquinas sin comprometer el rendimiento de clúster.
- ***Compatibilidad:*** Al trabajar con integración tradicional de datos y herramientas de análisis de datos.

Las tablas en Hive se asemejan a las tablas que se encuentran en una base de datos relacional, organizando las unidades de datos de menor a mayor granularidad. Las bases de datos están formadas por tablas que a su vez están formadas de particiones. Dentro de una base de datos particular, las tablas están serializadas y a cada tabla le corresponde una carpeta en el HDFS. Cada tabla puede ser subdividida para determinar la manera en que será distribuida, y a su vez esta subdivisión se verá reflejada en las sub-carpetas creadas.

Dentro de los tipos primitivos de datos que Hive soporta se encuentran: **a)** *BIGINT* **b)** *BINARY* **c)** *BOOLEAN* **d)** *CHAR* **e)** *DECIMAL* **f)** *DOUBLE* **g)** *FLOAT* **h)** *INT* **i)** *SMALLINT* **j)** *STRING* **k)** *TIMESTAMP* **l)** *TINYINT*

Estos tipos de datos primitivos pueden ser combinados formando estructuras de datos más complejas dando lugar a estructuras, mapas o arrays.

2.3.6. HBase

HBase es una base de datos columnar distribuida construida sobre HDFS. Su uso se justifica cuando se requiere lectura o escritura en tiempo real sobre grandes estructuras de datos. Las siguientes características justificarían la elección de HBase para el almacenamiento de datos semiestructurados como archivos de log y para el posterior acceso a esos datos de manera rápida [17].

Característica	Beneficio
Tolerancia a fallos	▪ ***Replicación*** en el centro de datos. ▪ ***Consistencia*** en las operaciones row-level. ▪ ***Alta disponibilidad*** mediante la recuperación automática de fallos. ▪ ***Balance automático de tablas***.
Rapidez	▪ ***Búsquedas cercanas a tiempo real***. ▪ ***Almacenamiento en caché***.
Usabilidad	▪ ***Variedad de modelos de datos*** para múltiples usos. ▪ ***Métricas*** a través de los plugins File y Ganglia. ▪ ***API Java*** de uso fácil así como APIs Thrift y REST.

CUADRO 2.1: Características y beneficios de HBase.

2.3.7. ZooKeeper

ZooKeeper es un servicio centralizado para el mantenimiento de la información de configuración, de sincronización distribuida y gestión de acceso a grupos. Permite la coordinación entre procesos distribuidos a través de un espacio jerárquico de nombres guardados en registros, llamados registers znodes. ZooKeeper ofrece a sus clientes un alto throughput, baja latencia, alta disponiblidad y acceso ordenado a los znodes, a diferencia de

la mayoría de sistema de ficheros. Este rendimiento permite a ZooKeeper ser utilizado en grandes sistemas distribuidos. Por otra parte, el mantenimiento del estricto orden permite sofisticados métodos de sincronización en los clientes [18].

ZooKeeper posee una interfaz y servicios sencillos donde se puede destacar su ***rapidez***, ya que está pensado para trabajar en entornos donde el ratio lectura/escritura sea de 10:1; ***fiabilidad***, gracias a la replicación a través de un grupo de hosts donde cada uno es consciente de los demás no permitiendo así ningún punto donde el sistema pueda colapsar por un fallo; ***simplicidad***, gracias al espacio de nombres ordenado jerárquicamente y ***orden***, responsabilidad del registro de todas las transacciones que se utiliza para abstracciones a más alto nivel, como sincronización de primitivos.

2.3.8. Apache Ambari

Apache Ambari es un plataforma de código libre orientada al aprovisionamiento, monitorización y control de la seguridad en clusters de Apache Hadoop. Ambari ofrece a los usuarios una interfaz Web y una API REST para facilitar las operaciones automáticas en el cluster. El uso de Ambari para manejar el cluster ofrece los siguientes beneficios [19]

- ***Fácil instalación, configuración y manejo***. Ambari simplifica el proceso de configuración gracias al uso de Smart Configs.
- ***Seguridad centralizada***. Reduciendo de esta forma la complejidad de administrar y configurar la seguridad del cluster.
- ***Visibilidad completa de la salud del cluster***. Para garantizar la salud del cluster Ambari ofrece la configuración de alarmas predefinidas y la visión de métricas de las operaciones ejecutadas en el cluster.
- ***Altamente personalizable***. Permite cuadrar Apache Hadoop en el entorno de trabajo. Se beneficia del uso de Ambari Stacks y de Ambari Views para aumentar el grado de personalización con la implementación de servicios adicionales propios a la interfaz web de Ambari.

Capítulo 3

Diseño

3.1. Arquitectura de un Centro de Datos

Para plantear el primer acercamiento al cluster y a su estructura se definirán los elementos que componen un centro de datos. Un centro de procesamiento de datos está compuesto principalmente por tres componentes básicos:

- ***Servidores.***
- ***Conexiones de red.***
- ***Sistemas de almacenamiento.***

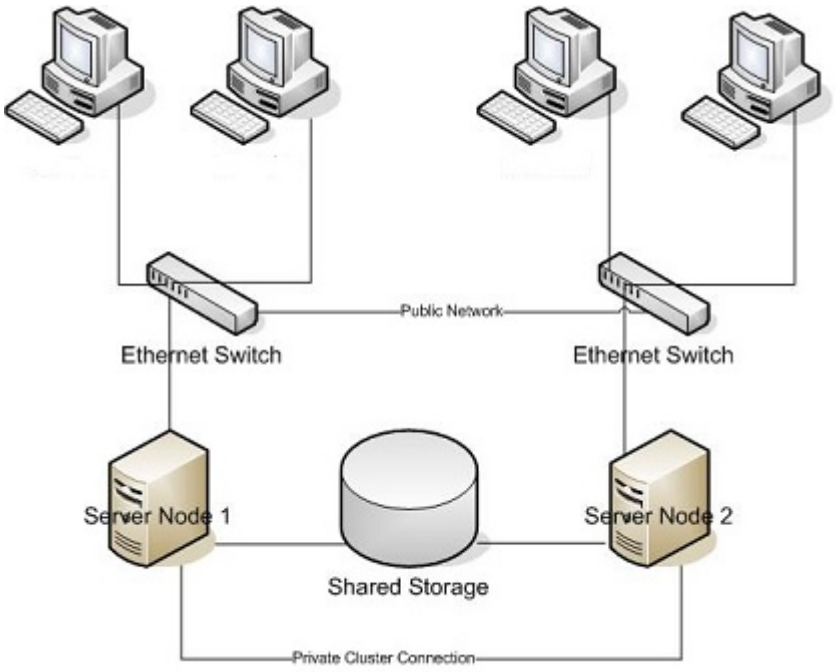


FIGURA 3.1: Diferentes componentes de la arquitectura en un cluster: servidores, switches y almacenamiento externo.

Los servidores serán los encargados de ejecutar los diferentes servicios y tareas, por tanto, son la base de la capacidad de cómputo del cluster. En centros de datos se suelen almacenar en racks. Por este motivo, más adelante se estudiará la emulación que ofrece Ambari en cuanto a racks virtuales.

Las conexiones de red entre los diferentes racks y los switches no determinan exclusivamente el rendimiento del cluster, pero sí pueden limitarlo. Si la configuración no es la apropiada se formarán cuellos de botella anulando la capacidad de cómputo de los servidores. Dentro de la estructura de red del cluster se puede diferenciar entre el backbone del cluster y las conexiones dentro de los racks. El backbone de la red comunica los racks y el objetivo será minimizar el tráfico en esta parte de la red.

Dado que el centro de datos estará orientado al análisis de Big Data, se necesitará un almacenamiento capaz de guardar cantidades de datos de terabytes o petabytes. Se utilizará un centro de almacenamiento de datos basado en los centros de datos de Google.

3.1.1. Servidores

La capacidad de cómputo del cluster vendrá determinada por los servidores que lo formen. Por tanto, esta dependerá de las características de las máquinas y de la configuración conjunta de ellas. Se dispondrá de N máquinas interconectadas entre ellas en M grupos, siendo M un divisor de N. Esta división hará referencia al número de máquinas de las que dispondrá cada rack. Siguiendo esta línea, esto significa que el cluster tendrá M racks. Aunque los racks no sean físicos, se establecerán racks virtuales como método de organización de las máquinas en el cluster. Al tener las máquinas distribuidas de esta forma, dentro de cada rack se podrán aumentar las réplicas de los datos sin aumentar el tráfico en el backbone de la red. Al tratarse de una emulación y no una configuración física, se podrá jugar con el número de máquinas por racks así como con el número de racks y evaluar el aumento del tráfico en la red. En cuanto al número de máquinas, el cluster será estático y no escalará; esto quiere decir que el número de máquinas será fijo y la gestión de recursos se realiza en el nivel de aplicación por Hadoop YARN. Esta será una diferencia sustancial con sistemas de computación distribuida en la nube, donde el número de máquinas escala conforme a la demanda.

Junto a los servidores organizados en racks que trabajarán como esclavos, encontraremos un master que controlará la actividad del cluster. Este master es consciente de todas las actividades que se llevan en el cluster así como todos los datos almacenados en él, por tanto, ejecutará todos los servicios de monitorización y control de las diferentes aplicaciones que se ejecuten en el cluster.

Requisitos de los servidores

Dependiendo del uso que se le vaya a dar a un determinado cluster, se puede optar por diferentes opciones de máquinas, cada grupo con unas características optimizadas a los requerimientos. Por ejemplo, si la aplicación va a necesitar un alto rendimiento gráfico como en streaming de aplicaciones en 3D, aprendizaje automático o codificación de vídeo, se puede optar por un grupo de instancias con GPU de alto rendimiento. Si se van a realizar un uso intensivo de trabajo sobre la memoria RAM como podría ser con Apache Spark, se pueden elegir instancias optimizadas para ejecutar aplicaciones en memoria a gran escala. En este caso, dado que no se conoce

la utilidad final del cluster, se optará por instancias de carácter más general, que ofrezcan un equilibrio entre las necesidades informáticas, de memoria y de gráficos y que a su vez garanticen la solvencia del cluster en la ejecución de cualquier tarea gracias al beneficio de la distribución de tareas.

3.1.2. Almacenamiento externo

Requisitos del almacenamiento externo

Cuando el almacenamiento de los servidores no es suficiente, o no es óptimo el guardar los datos en estos servidores, se recurre a formas de almacenamiento externo donde guardar los datos y poder operar con ellos cuando sea necesario. Estos sistemas de almacenamiento por tanto son ajenos a los servidores y se pueden ver como un componente más del cluster.

Una posible configuración de un cluster que cuente con un sistema de almacenamiento sería montar un sistema de ficheros distribuido entre los nodos del cluster, de tal forma que no se requieran nodos con gran capacidad de almacenamiento a nivel local. A pesar de esta limitación, al agregar la capacidad de almacenamiento se podría conseguir un sistema de ficheros distribuido suficiente para el objetivo del análisis de Big Bata. En el almacenamiento externo se almacenarían de manera local los datos y cuando fuese necesario trabajar sobre ellos se subirían al sistema de ficheros. Siguiendo este modelo y la finalidad docente de este proyecto, esta sería una buena aproximación para solventar el problema del alojamiento de los datos.

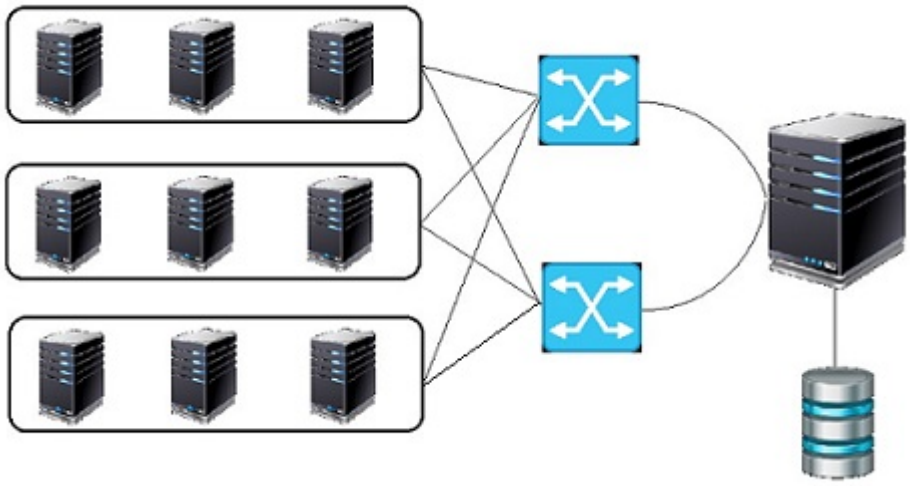


FIGURA 3.2: Diferentes componentes de la arquitectura en un cluster: servidores, switches y almacenamiento externo.

3.1.3. Switches

Los switches elegidos para este diseño serán los Cisco Small Business SG200-18 [20]. Estos conmutadores pertenecen a la serie Cisco 200 de conmutadores inteligentes. Las principales ventajas que ofrecen frente a competidores son:

- ***Facilidad de configuración y manejo:*** Dispone de interfaces para el usuario a la hora de lanzar la configuración de la red, así como para el mantenimiento y la resolución de problemas.
  - *LLDP-MED:* El Cisco Discovery Protocol and Link Discovery Protocol detecta automáticamente los dispositivos conectados a la red y configuran dichos dispositivos con los parámetros apropiados de calidad de servicio.
  - *Cisco Smartports:* Esta tecnología permite complementar la configuración automática con la manual para definir la configuración de los puertos, en relación a la seguridad, calidad de servicio y disponibilidad según el tipo de dispositivo conectado a cada puerto.
  - *Cisco FindIt Network Discovery Utility:* Barra de herramientas que aparece en el navegador de un usuario conectado mostrando los demás dispositivos conectados a esa red así como cierta información sobre ellos, como su dirección IP.
- ***Rendimiento y alta fidelidad.***
- ***Alimentación mediante Internet (PoE):*** La alimentación mediante Internet está soportada tanto por la conexión Fast Ethernet como por Gigabit Ethernet facilitando la implementación al permitir mandar por el mismo canal físico datos y energía.
- ***Seguridad:*** Ofrece seguridad añadiendo el IEEE 802.1X port security para controlar el acceso a la red y prevención contra ataques de denegación de servicio (DoS).
- ***Soporte de VoIP:*** Capacidad para configurar parámetros de QoS con el fin de priorizar determinado tráfico más sensible a retrasos, permitiendo así las llamadas de voz sobre IP.
- ***Soporte del nuevo estándar IPv6:*** Se podrá usar el sistema de direcciones sin necesidad de cambiar el equipo al tener este soporte para este nuevo protocolo.

En cuanto al rendimiento de este switch, el SG200-18 ofrece una capacidad de conmutación de 26.78 en millones de paquetes de 64 bytes por segundo. Si se quiere dar el dato de la capacidad de conmutación en Gigabits por segundo (Gbps), esta es de 36 Gbps.

3.2. Disposición del cluster

A la hora de determinar la estructura de nuestro cluster, se tomará como escenario base uno de los laboratorios del departamento de ingeniería telemática. Aplicando la estructura que se conoce al cluster, esta es, un master y varios nodos esclavos en cuanto a servidores, esta es la disposición [21].

Se contará con 30 servidores que ejercerán de nodos esclavos. Estos nodos se organizarán en racks por filas: cada rack (virtual) constará de 5 máquinas. El master será el ordenador monitor01 en la figura 3.4.

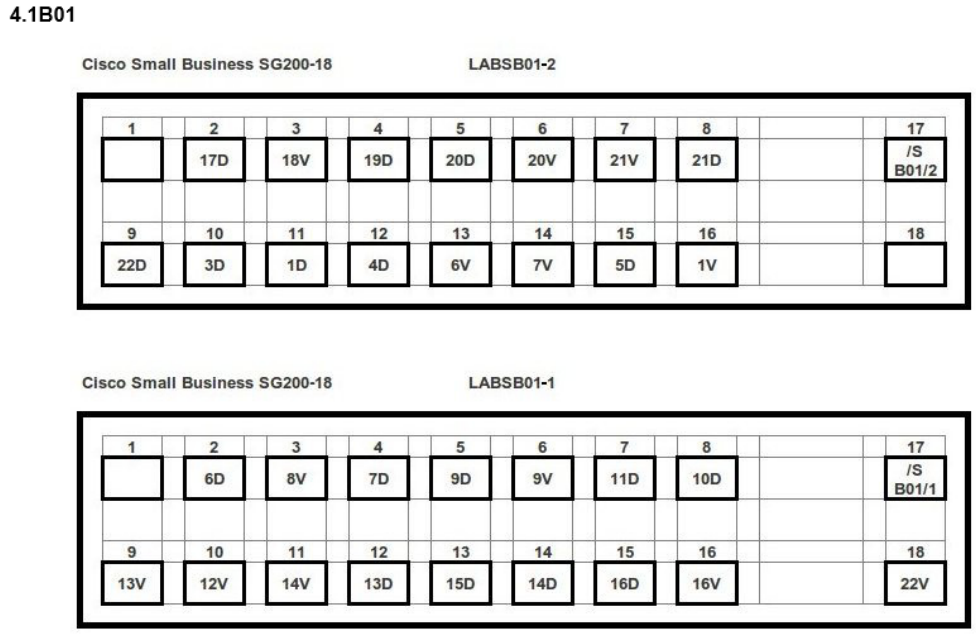


FIGURA 3.3: Conexión de las máquinas del laboratorio en el switch Cisco Small Bussiness SG200-18.

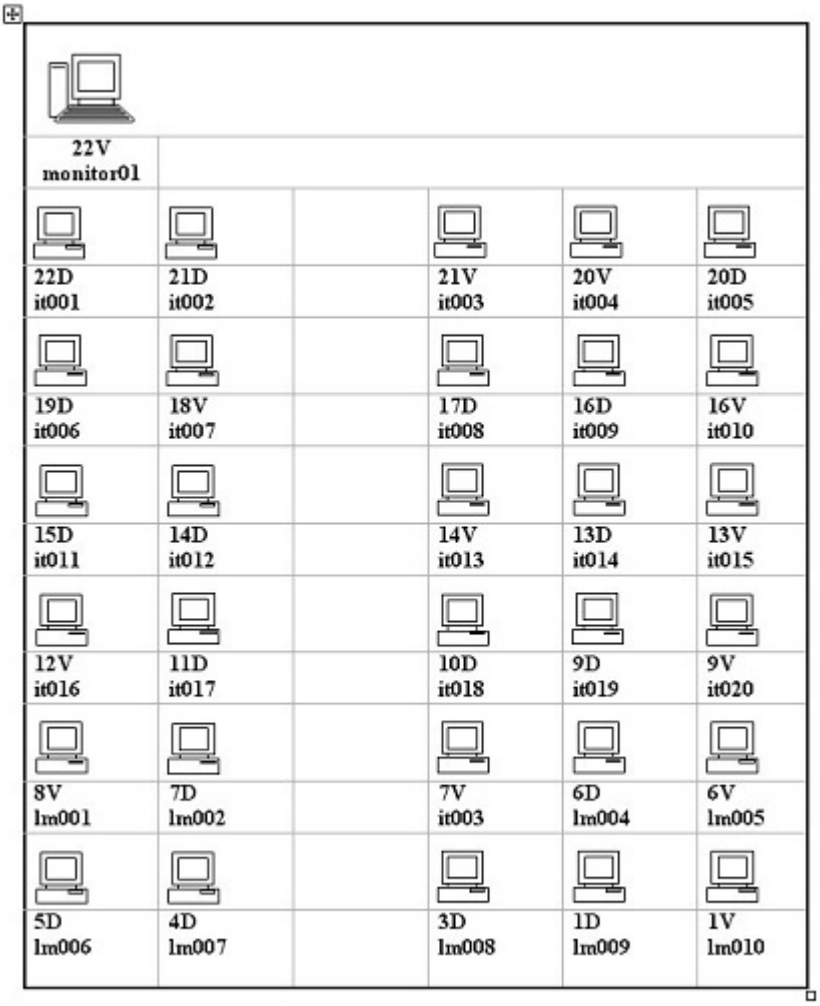


FIGURA 3.4: Organización del laboratorio en el que estará basado el diseño de nuestro cluster.

3.2.1. Componentes Cluster

En primer lugar, se deben conocer los elementos que forman la arquitectura de un cluster. El master ejecutará el servidor. Este master servirá de Name Node y JobTracker. Por otra parte, los esclavos servirán de Data Nodes y TaskTracker.

El núcleo de la arquitectura de un cluster en cuanto a servicios consta de [15]:

- ***HDFS.***
- ***MapReduce.***

En un cluster con los servicios básicos de HDFS y MapReduce se encuentran los siguientes componentes:

- ***NameNode***: Encargado del manejo del sistema de ficheros y del control del acceso. En cada cluster tiene que haber exactamente un NameNode.
- ***Secondary NameNode***: La función de este Name node es guardar puntos de control del actual NameNode para proveer de tolerancia a fallos al cluster. Al igual que ocurre con el NameNode, existe un Secondary NameNode para cada cluster.
- ***JobTracker***: Distribuye tareas a los nodos esclavos. En cada cluster Hadoop se encuentra un JobTracker.
- ***DataNode***: Guarda datos en el sistema de ficheros. Cada DataNode maneja de manera local los datos que alberga y guarda copias de algunos datos almacenados en otros DataNodes del cluster. En un cluster se pueden encontrar uno o más DataNodes.
- ***TaskTracker***: Esclavo encargado de ejecutar las tareas de Map y Reduce. En un cluster puede haber más de un TaskTracker.

El HDFS y MapReduce forman los dos servicios más básicos en un cluster para el análisis de Big Data. Con el fin de aumentar la funcionalidad del cluster, se implementarán otros servicios descritos con anterioridad, como son Spark, YARN, Hive, HBase o Pig.

Llegado el punto de agrupar todos los servicios, la opción que se presenta más razonable es el uso de Apache Ambari para el manejo, la monitorización y el control del cluster. A su vez, Ambari permite la instalación de los servicios desde interfaz web. La estructura que se seguirá será la instalación de un Ambari Server en el master del cluster, y los esclavos ejecutarán los clientes de Ambari.

3.2.2. Arquitectura de Ambari

Los componentes principales de Ambari son:

- ***Agente de Ambari***: Sistema de monitorización de Ganglia (gmond) instalado en cada nodo para la colección de métricas.

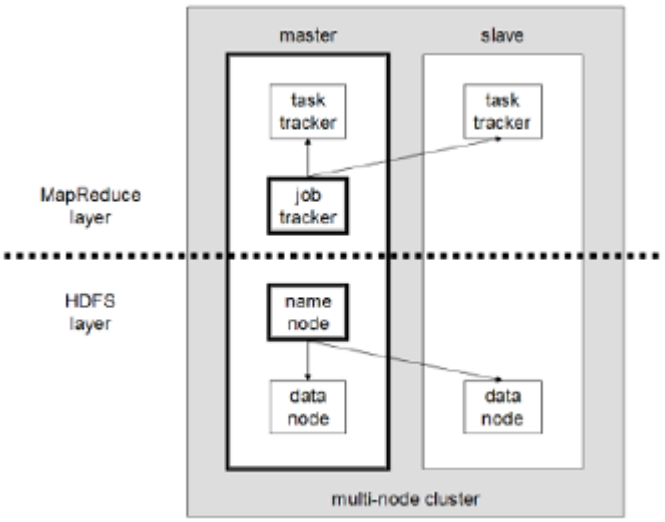


FIGURA 3.5: Arquitectura del NameNode y DataNode en un cluster multinodo.

- ***Servidor de Ambari***: Encargado de recoger la información emitida por los agentes de Ambari. El servidor de Ambari consta de los siguientes componentes:
  - ***Postgres RDBMS***: Sistema de manejo de base de datos para almacenar la configuración del cluster.
  - ***Servicio de Autenticación***: Permite la integración con un servicio de autenticación externo como LDAP.
  - ***Servicio Nagios***: Para añadir soporte a alertas.
  - ***APIs REST***: Para la integración con Ambari web pudiendo ser utilizada por aplicaciones externas personalizadas.

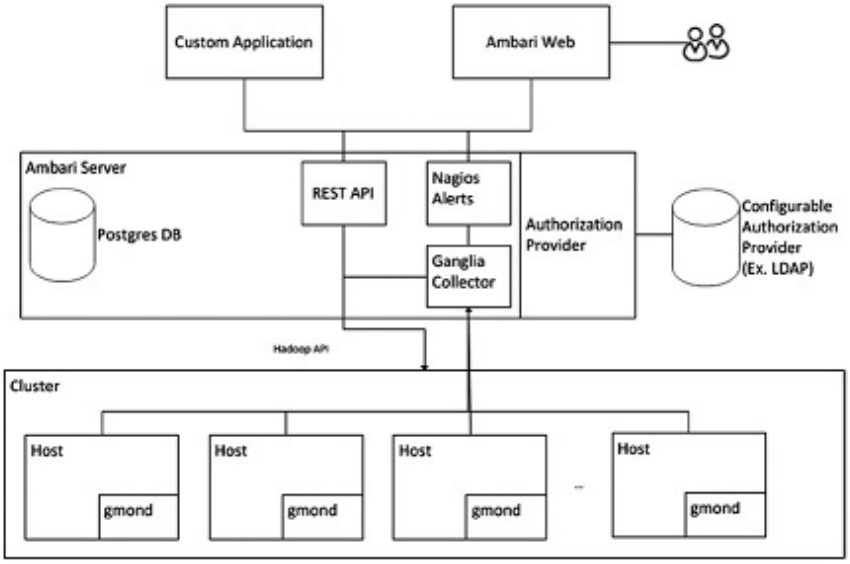


FIGURA 3.6: Diferentes niveles en la arquitectura de Ambari.

Capítulo 4

Prototipo

Debido a la escasez de recursos tanto humanos como físicos, y añadiendo la limitación temporal, el diseño inicial del cluster se ha adaptado a un prototipo factible de realizar con los recursos disponibles. Para ello, se han diferenciado dos prototipos. Su realización en el tiempo ha sido análoga a su aparición en este documento, por tanto el primer prototipo coincide con el primero en realizarse.

4.1. Primer prototipo

Para este primer prototipo se contará con dos máquinas con kernels de 64 bits. Sus nombres completos de dominio (FQDN) son:

- ***carissimi.gast.it.uc3m.es***
- ***frescobaldi.gast.it.uc3m.es***

Inicialmente se añadió una tercera máquina virtual para poder contar con dos esclavos y un master. Debido a la incapacidad de tener la máquina virtual conectada a la misma red que ambas máquinas y temiendo que eso supusiera una alteración en las posteriores métricas de tráfico en red, se optó por descartar esta configuración y montar en el primer prototipo únicamente las dos máquinas mencionadas. Se decidió debido a la mayor

potencia de *carissimi* que fuese esta máquina la encargada de ejecutar al servidor de Ambari. Mencionar en este punto que al seguir el proceso de instalación del servidor, el master que sirve de host a este servidor se configura automáticamente como agente; por tanto siguiendo esta estructura se tendría un servidor y un agente en *carissimi* y un agente en *frescobaldi*. Dos máquinas, un servidor y dos agentes.

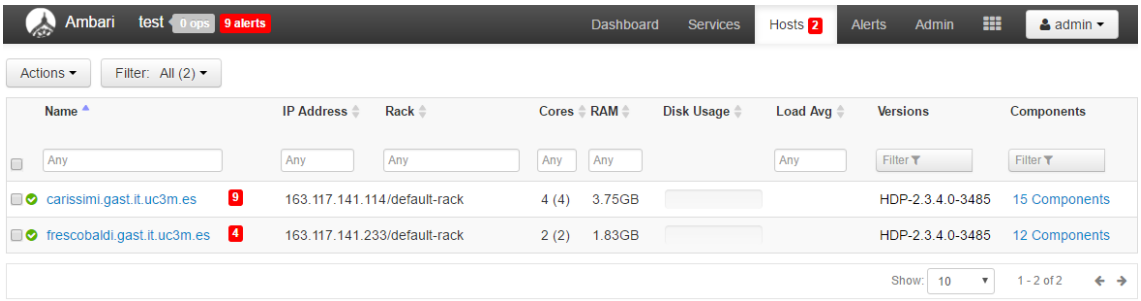


FIGURA 4.1: Información que muestra la interfaz web de Ambari sobre los dos hosts que forman el cluster.

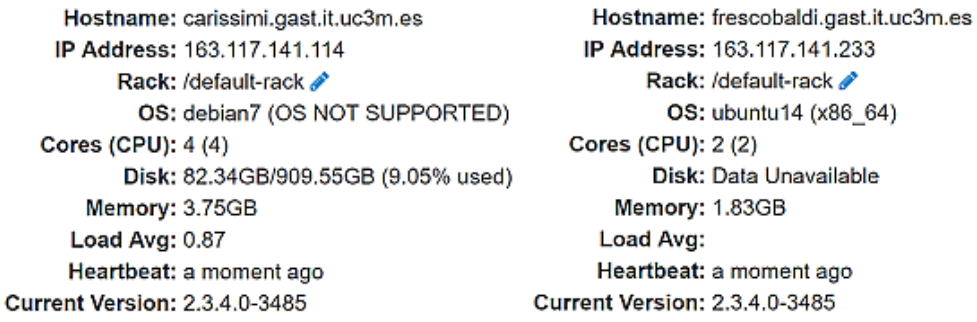


FIGURA 4.2: Información adicional de los hosts del Prototipo 1.

4.1.1. Componentes Instalados

A la hora de determinar los servicios y componentes instalados, se diferenciarán por host, por un lado se estudiarán los servicios instalados en el master *carissimi* y por el otro, los instalados en el agente *frescobaldi*. Los componentes instalados en el servidor, *carissimi*, son los que se ven en la figura 4.3.

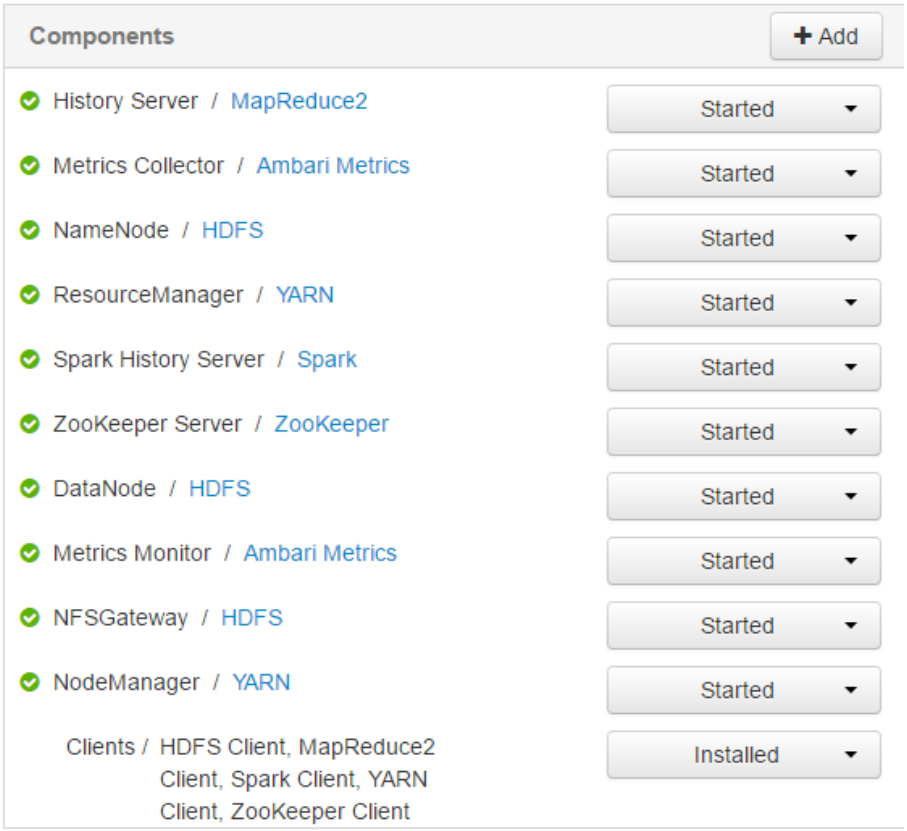


FIGURA 4.3: Información y estado de los componentes instalados en carissimi.

Los componentes instalados en frescobaldi se muestran en la figura 4.4.

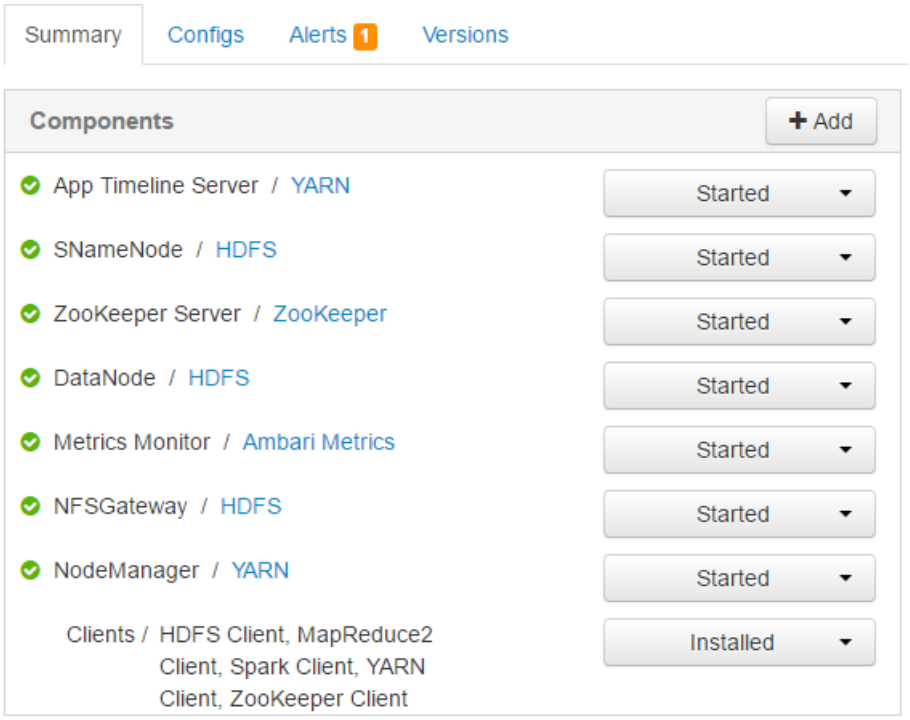


FIGURA 4.4: Información y estado de los componentes instalados en frescobaldi.

4.1.2. Descripción de los componentes

Primero se describirán aquellos componentes instalados en el servidor, *carissimi*, que no están presentes en el esclavo, *frescobaldi*. Estos componentes son el *History Server* de MapReduce, el *Metrics Collector* de Ambari Metrics, el *Name Node* del HDFS, el *Resource Manager* de YARN y el *Spark History Server* de Spark.

History Server de MapReduce

El *History Server* de MapReduce proporciona una API con información sobre las tareas ejecutadas en MapReduce. En la interfaz de Ambari, desde el servicio de MapReduce se puede acceder a links rápidos para obtener la siguiente información:

- *JobHistory UI*
- *JobHistory Logs*
- *JobHistory JMX*
- *Thread Stack*

Metrics Collector de Ambari

El *Metrics Collector* de Ambari es un daemon que recibe la información de los *Ambari Monitors* y de los *Metrics Hadoop Sinks*. El sistema de recolección de métricas puede guardar los datos bien en el propio almacenamiento local del servidor o bien en el sistema de ficheros distribuido del cluster en el que esté ejecutándose. Este daemon está construido sobre tecnologías Hadoop como son HBase Phoenix.

Se pueden encontrar especificaciones sobre la API de este servicio al ser código libre. Los dos tipos de peticiones que puede recibir el *Metrics Collector* son de tipo POST y GET [22].

La petición POST es utilizada para mandar métricas al servicio de Ambari Metrics. El Ambari Sink guarda en un buffer información durante 1 minuto para después mandar esta información al AMS. La manera de realizar la petición sería la siguiente:

POST *http://<ambari-metrics-collector>:6188/ws/v1/timeline/metrics*

```
{
  "metrics": [
    {
      "metricname": "AMBARI_METRICS.SmokeTest.FakeMetric",
      "appid": "amssmoketestfake",
      "hostname": "ambari20-5.c.pramod-thangali.internal",
      "timestamp": 1432075898000,
      "starttime": 1432075898000,
      "metrics": {
        "1432075898000": 0.963781711428,
        "1432075899000": 1432075898000
      }
    }
  ]
}
```

FIGURA 4.5: Estructura enviada al realizar una petición POST al *Metrics Collector*.

La respuesta obtenida a una petición GET sería la siguiente.


```
{
   "metrics": [
      {
         "timestamp": 1432075898089,
         "metricname": "AMBARI_METRICS.SmokeTest.FakeMetric",
         "appid": "amssmoketestfake",
         "hostname": "ambari20-5.c.pramod-thangali.internal",
         "starttime": 1432075898000,
         "metrics": {
            "1432075898000": 0.963781711428,
            "1432075899000": 1432075898000
         }
      }
   ]
}
```


FIGURA 4.6: Estructura recibida al realizar una petición GET al *Metrics Collector*.

NameNode de HDFS

El NameNode [23] es la pieza central del sistema distribuido de ficheros de Hadoop. Se encarga de guardar el árbol de los directorios de todos los archivos que están almacenados y saber en qué nodo se encuentran estos archivos. Los directorios y los archivos se representan mediante los llamados *inodes*, que guardan un registro de diferentes atributos, como permisos, modificaciones y tiempos de acceso o espacio ocupado en disco. Cada archivo resulta dividido típicamente en bloques de 128 Megabytes, aunque el usuario puede determinar este parámetro para cada archivo, y estos bloques se guardan en réplicas a lo largo de los DataNodes. De esta distribución es de la que se encarga el NameNode.

Cuando un cliente HDFS quiere leer un archivo contacta con el NameNode para obtener la localización del bloque o bloques que contiene esa información pedida. Una vez que obtiene la localización ya se contactaría con el DataNode más cercano al cliente, debido a la posible variedad en la respuesta por las réplicas. Si el cliente quisiera escribir un archivo, pediría

al NameNode un grupo de DataNodes donde poder guardar el archivo. El número de DataNodes en la respuesta dependerá del valor de replicación que se tenga configurado. Por defecto, este valor es de 3. Como ya se ha descrito en la arquitectura de un cluster Hadoop, según el diseño actual un cluster contiene un Name Node y un número variable de DataNodes. En este caso, se contará con un NameNode (*carissimi*) y dos DataNodes (*carissimi* y *frescobaldi*).

HDFS guarda todo el árbol de directorios y archivos en memoria, así como la información de los bloques que pertenecen a cada archivo, llamada imagen. El NameNode guarda checkpoints de estas imágenes así como ficheros de log con las modificaciones.

En el host *carissimi* la interfaz web del NameNode es accesible en el puerto 50070. La información que ofrece Ambari sobre el componente general de HDFS es la siguiente, además de unas métricas proporcionadas por el sistema de métricas.

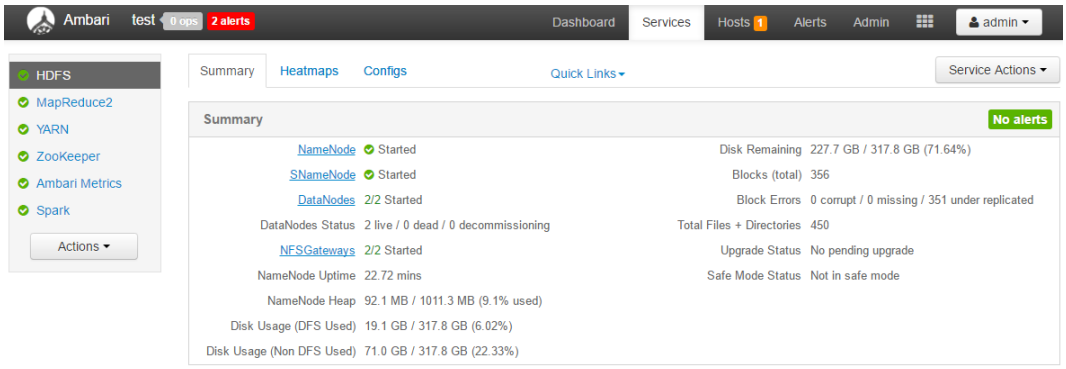


FIGURA 4.7: Información que ofrece Ambari sobre HDFS: NameNode, SNameNode, DataNodes entre otros.

La interfaz web del NameNode ofrece información general del sistema de ficheros como la capacidad, la cantidad de DFS utilizada, la no utilizada; sobre los nodos del cluster, sobre los bloques y sus réplicas, entre otra información en su pestaña Overview.

Dentro de las utilidades, se encuentran los logs y la opción de un explorador de ficheros con interfaz gráfica. Esta opción es de gran utilidad, ya que la manera de gestionar el HDFS es mediante el terminal y puede resultar algo abstracto para aquellos usuarios que no estén muy acostumbrados. Desde este explorador se puede obtener una imagen clara y concisa de los diferentes archivos y directorios que se alojan en el HDFS así como las partes de cada archivo y su ubicación.

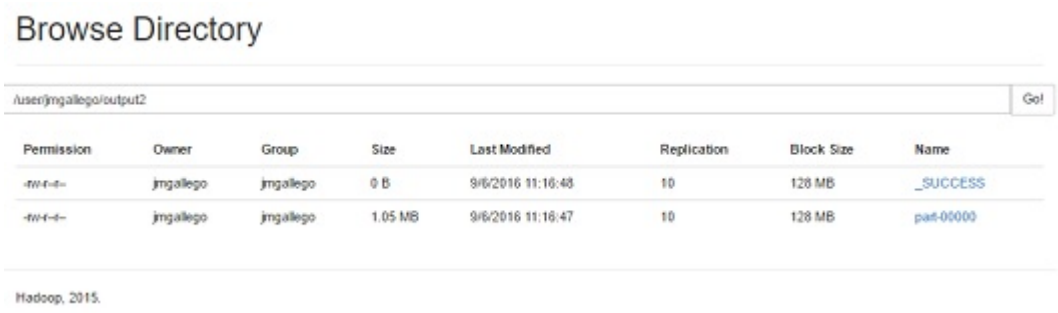


FIGURA 4.8: Ejemplo del explorador de archivos del NameNode. Se aprecia el output de una prueba realizada en el fichero *part-00000*.

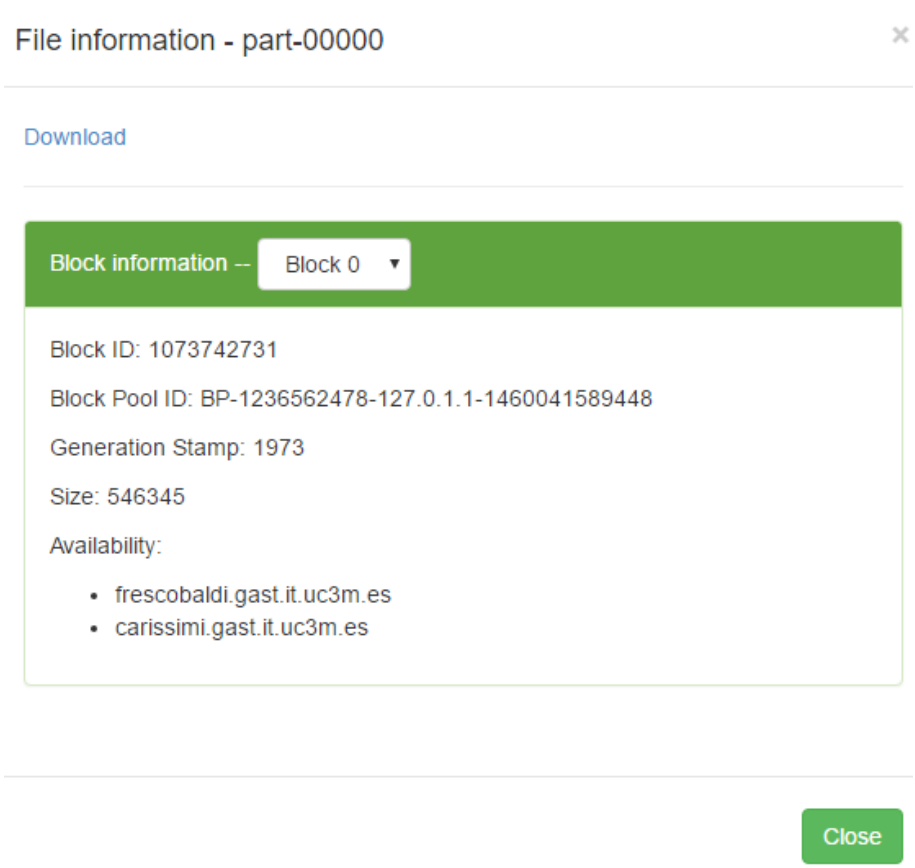


FIGURA 4.9: Información que muestra el explorador gráfico sobre el fichero *part-00000*. Se puede obtener el ID, el tamaño, su disponibilidad en los diferentes nodos del cluster e incluso la descarga directa.

En la opción utilities logs, se puede consultar los audit.log ordenados por orden cronológico. La ruta para acceder a los logs sería la siguiente: *http://<nombre del servidor>:50070/logs/*

Resource Manager de YARN

El Resource Manager de YARN es el encargado de manejar y controlar todos los recursos disponibles en el cluster. Trabaja de manera conjunta con el Node Manager presente en cada nodo y con el Application Master de cada aplicación [24].

El Resource Manager está constituido por los siguiente componentes:

- ***ClientService:*** Se trata de la interfaz al cliente encargada de controlar todas las interfaces RPC del cliente al Resource Manager, cubriendo las operaciones de subir una aplicación, terminar una aplicación, obtención de información de la cola y estadísticas del cluster, entre otras. La interfaz gráfica que obtenemos de nuestro RM alojado en *carissimi* se puede ver en la figura 4.12.
- ***AdminService:*** Se encarga de asegurar que el cluster no se queda sin recursos debido a excesivas peticiones de clientes, de organizar las

prioridades, de las tareas relacionadas con la administración del sistema como son actualizar la lista de nodos, configuración de colas y demás operaciones.

- ***ResourceTrackerService:*** Es el componente que se encarga de comunicarse mediante RPC con todos los nodos controlados por YARN. Registra nuevos nodos, acepta o rechaza conexiones de nodos inválidos, controla los heartbeats de los nodos para hacer el seguimiento de la salud del cluster y enviarlos al YarnScheduler (programador). Trabaja conjuntamente con el NMLivelinessMonitor y con el NodesListManager.

- ***NMLivelinessMonitor:*** Encargado de asegurar los nodos que están activos y dar de baja en el RM aquellos que no responden a los heartbeats en un tiempo predefinido por defecto de 10 minutos. Si esto sucede, ese nodo se marca como muerto y no se le asignan más tareas.

- ***NodesListManager:*** Colección de nodos válidos y excluidos. Genera esta lista de nodos a partir de los ficheros de configuración de cada nodo, que se encuentran en las dos siguientes rutas: *yarn.resourcemanager.nodes.include-path* y *yarn.resourcemanager.nodes.exclude-path*.

- ***ApplicationMasterService:*** Componente encargado de comunicarse vía RPC con los AM. Se encarga del registro de nuevos AM así como de la terminación de estos, de las peticiones de reserva para los AMs que se están ejecutando y pasarlos al YarnScheduler.

- ***AMLivelinessMonitor:*** Guarda el estado de cada AM y su último heartbeat para ayudar a manejar la lista de AMs.

- ***ApplicationsManager:*** Mantiene la colección de las aplicaciones. Guarda información de estas aplicaciones por si el cliente la requiere vía interfaz web o vía terminal una vez terminada la aplicación.

- ***ApplicationACLsManager:*** Componente encargado de mantener las listas ACL de cada aplicación.

- ***ApplicationMasterLauncher:*** Conserva una piscina de threads para las aplicaciones de nuevo lanzamiento y para aquellas que por algún motivo requieran de otro Application Manager. Cuando una aplicación termina ya sea de manera satisfactoria o forzada, será el encargado de eliminar el Application Manager de dicha aplicación.

- ***YarnScheduler:*** Encargado de asignar los recursos a las aplicaciones que se estén ejecutando teniendo en cuenta los requisitos y las restricciones del sistema. Organiza las colas estableciendo prioridades en función de memoria requerida, CPU, espacio en disco, requerimiento de red, entre otros.

- ***ContainerAllocationExpirer:*** Mantiene una lista de los AMs cuyo NM no ha reportado que estén ejecutándose. Si pasado un intervalo predefinido por defecto de 10 minutos no se ha recibido esa confirmación, el AM sería eliminado por el RM.

- ***TokenSecretManagers:*** El ResourceManager mantiene una colección de SecretManagers que almacenan tokens de seguridad para autenticar o autorizar peticiones RPC. Se pueden diferenciar los siguientes managers:
  - ***ApplicationTokenSecretMAnager.***
  - ***ContainerTokenSecretMAnager.***
  - ***RMDelegationTokenSecretManager.***
  - ***DelegationTokenRenewer.***

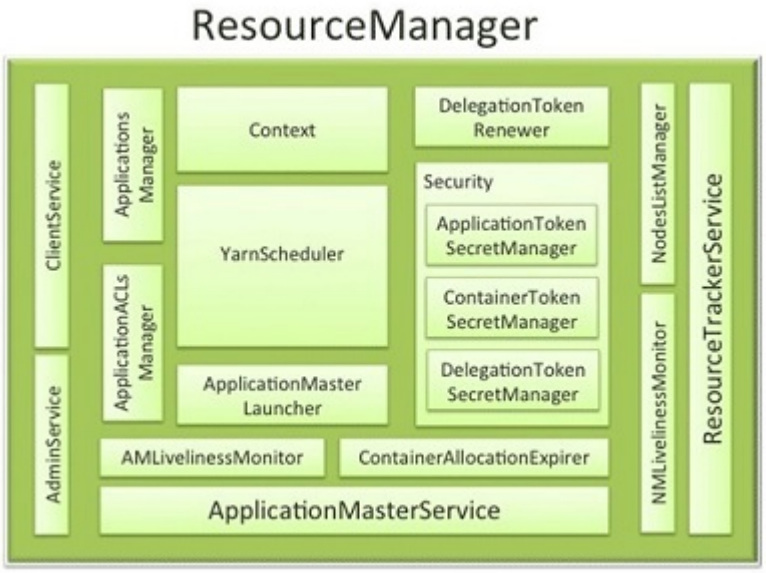


FIGURA 4.10: Descripción de los componentes que forman el Resource Manager de YARN.

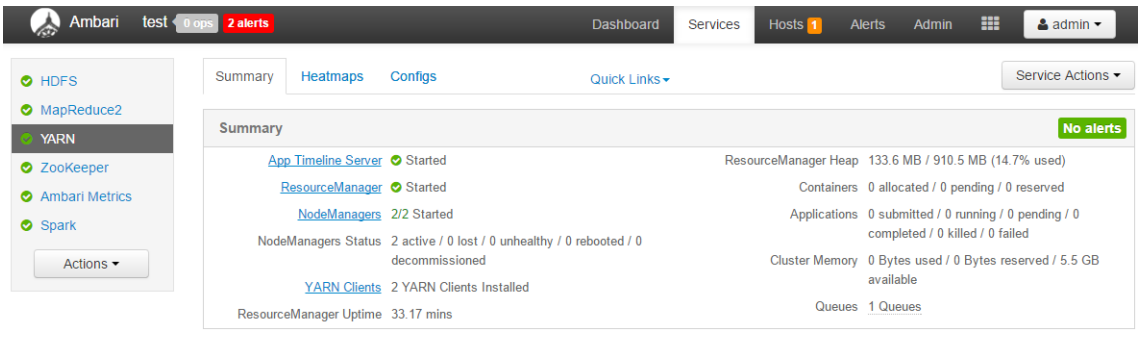
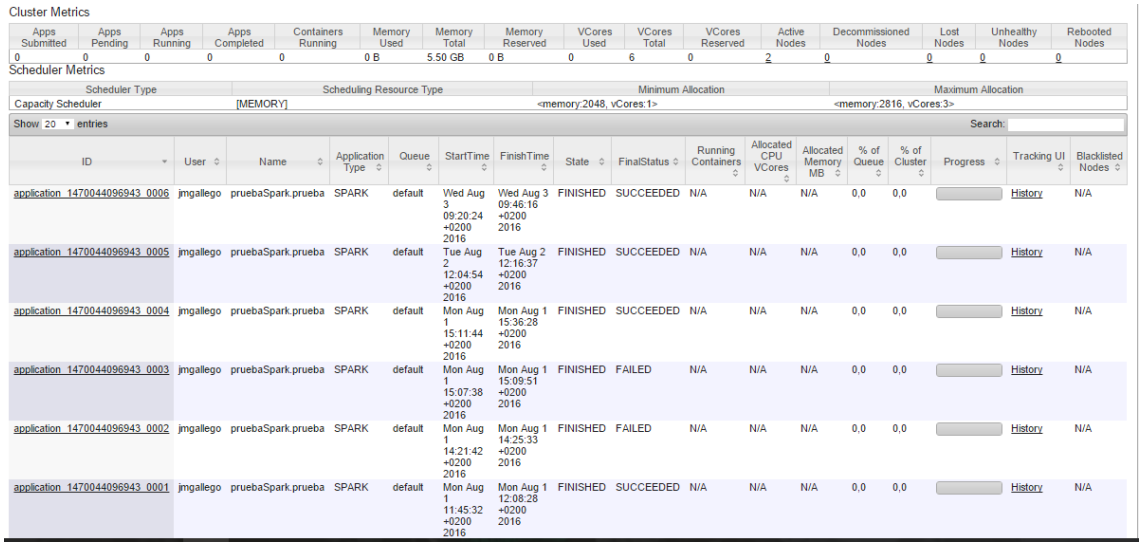


FIGURA 4.11: Interfaz Web de usuario de YARN que ofrece Ambari con acceso a un resumen, configuraciones y a la interfaz propia del RM.

Ambari ofrece un link para acceder de manera intuitiva al Resource Manager, alojado en el puerto 8088.Como se puede ver en la figura 4.11, esta interfaz gráfica ofrece información general del cluster. Se podrá obtener información de los nodos que forman el cluster en el apartado *Nodes*, figura



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
0	0	0	0	0	0 B	5.50 GB	0 B	0	6	0	2	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:2048, vCores:1>	<memory:2816, vCores:3>

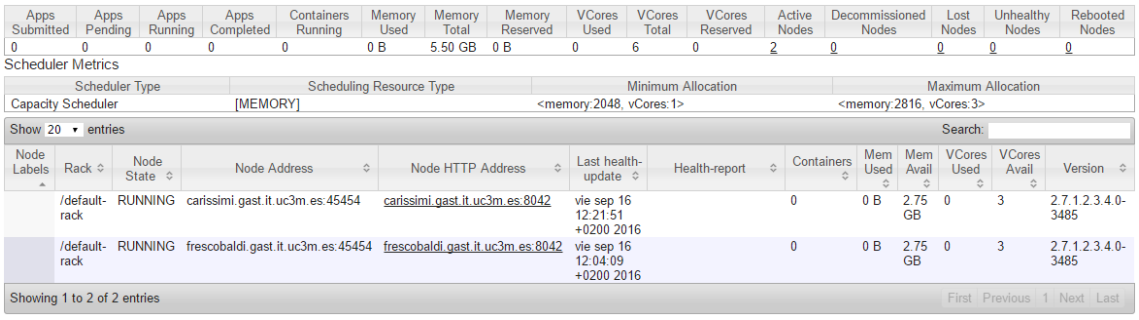
Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1470044096943_0006	jmgallego	pruebaSpark.prueba	SPARK	default	Wed Aug 3 09:20:24 +0200 2016	Wed Aug 3 09:46:16 +0200 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	0,0	0,0		History	N/A
application_1470044096943_0005	jmgallego	pruebaSpark.prueba	SPARK	default	Tue Aug 2 12:04:54 +0200 2016	Tue Aug 2 12:16:37 +0200 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	0,0	0,0		History	N/A
application_1470044096943_0004	jmgallego	pruebaSpark.prueba	SPARK	default	Mon Aug 1 15:11:44 +0200 2016	Mon Aug 1 15:36:28 +0200 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	0,0	0,0		History	N/A
application_1470044096943_0003	jmgallego	pruebaSpark.prueba	SPARK	default	Mon Aug 1 15:07:38 +0200 2016	Mon Aug 1 15:09:51 +0200 2016	FINISHED	FAILED	N/A	N/A	N/A	0,0	0,0		History	N/A
application_1470044096943_0002	jmgallego	pruebaSpark.prueba	SPARK	default	Mon Aug 1 14:21:42 +0200 2016	Mon Aug 1 14:25:33 +0200 2016	FINISHED	FAILED	N/A	N/A	N/A	0,0	0,0		History	N/A
application_1470044096943_0001	jmgallego	pruebaSpark.prueba	SPARK	default	Mon Aug 1 11:45:32 +0200 2016	Mon Aug 1 12:08:28 +0200 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	0,0	0,0		History	N/A

FIGURA 4.12: Interfaz Web de usuario del Resource Manager de YARN alojado en el puerto 8088 de carissimi.

4.13. Se observa que los dos nodos del cluster están corriendo alojados en el mismo rack virtual que se crea por defecto. Se dispone también de las direcciones de los nodos así como las direcciones HTTP, información sobre el último chequeo de salud, memoria RAM disponible en cada nodo y núcleos virtuales disponibles.

También se observa información general del cluster de forma agregada: se cuentan con 5.5 GB de memoria RAM y 6 núcleos virtuales repartidos entre los dos nodos.



Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
0	0	0	0	0	0 B	5.50 GB	0 B	0	6	0	2	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:2048, vCores:1>	<memory:2816, vCores:3>

Show 20 entries Search:

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail	Version
	/default-rack	RUNNING	carissimi.gast.it.uc3m.es:45454	carissimi.gast.it.uc3m.es:8042	vie sep 16 12:21:51 +0200 2016		0	0 B	2.75 GB	0	3	2.7.1.2.3.4.0-3485
	/default-rack	RUNNING	frescobaldi.gast.it.uc3m.es:45454	frescobaldi.gast.it.uc3m.es:8042	vie sep 16 12:04:09 +0200 2016		0	0 B	2.75 GB	0	3	2.7.1.2.3.4.0-3485

Showing 1 to 2 of 2 entries First Previous 1 Next Last

FIGURA 4.13: Información que ofrece el RM sobre los Nodos que forman el cluster.

Spark History Server

El último componente instalado exclusivamente en el master es el Spark History Server. En los clusters donde Spark se ejecuta en modo Standalone, esto es, trabaja directamente sobre el sistema de ficheros y no hace uso de un gestor de recursos, como YARN, Spark se encarga de gestionar los recursos de forma independiente. De manera análoga al ResourceManager, el History Server de Spark proporciona una interfaz web donde consultar diversa información sobre el cluster y sobre las aplicaciones ejecutadas.

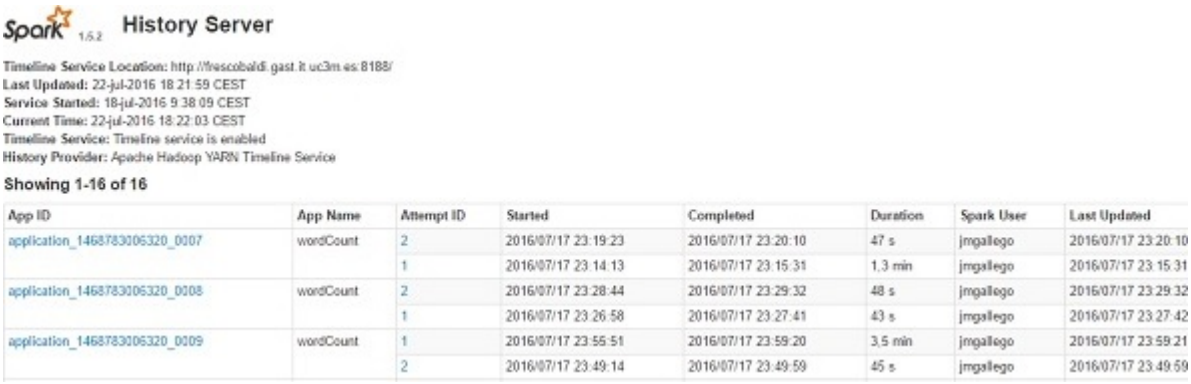


FIGURA 4.14: Información que ofrece el History Server de Spark sobre las aplicaciones ejecutadas.

Los siguientes son los componentes que están presentes tanto en *carissimi* como en *frescobaldi*. Esto significa que son los componentes que los agentes de Ambari ejecutan.

Node Manager de YARN

El Node Manager de YARN es el componente que se encarga de las tareas de monitorización y control individualmente de cada nodo. Monitoriza actividad de la CPU del nodo, de los contenedores de las diferentes aplicaciones, se comunica con el Resource Manager, controla que el nodo se encuentre activo, gestiona los logs y se encarga de otros servicios que pueden ser utilizados por YARN [25].

Al igual que para el Resource Manager, se van a explicar los componentes que forman el Node Manager, que son los de la figura 4.15.

- ***NodeStatusUpdater:*** La primera vez que se levanta el nodo, este componente se comunica con el ResourceManager para informar de los recursos disponibles. En las posteriores comunicaciones con el RM, ofrece actualizaciones del estado del nodo. El RM también se puede comunicar con este componente para terminar algún contenedor que se encuentre ejecutándose en este nodo.
- ***ContainerManager:*** Componente formado por varios subcomponentes, forma el núcleo del NodeManager.

  - ***Servidor RPC:*** Es el encargado de atender las conexiones de los Application Masters con el fin de levantar nuevos contenedores o terminar alguno que se encuentre ejecutándose. Para garantizar la seguridad, la autenticación y la autorización de las comunicaciones trabaja con el ContainerTokenSecretManager. Las operaciones quedan registradas en un log que puede ser consultado posteriormente por servicios de seguridad, el audit-log.
  - ***ResourceLocalizationService:*** Encargado de descargar y distribuir archivos con información de los recursos que más adelante los contenedores pueden utilizar.
  - ***ContainersLauncher:*** Este componente guarda una serie de threads para poder lanzar los contenedores lo más rápido posible. Además de lanzar los procesos, también los termina a petición del RM o de los AMs.
  - ***AuxServices:*** Módulo que permite la posibilidad de implementar servicios auxiliares adicionales. Se debe tener en cuenta que estos servicios deben configurarse antes de que el NodeManager esté corriendo.
  - ***ContainersMonitor:*** Una vez un contenedor es lanzado y se encuentra ejecutándose, este componente se encarga de monitorizarlo y comprobar que hace uso de los recursos que le han sido reservados. Si este no fuera el caso y estuviera utilizando más recursos, este componente puede terminar con el contenedor. De esta forma se garantiza el reparto justo de recursos entre los diferentes contenedores y se evita que un contenedor con un mal uso afecte a los demás contenedores.
  - ***LogHandler:*** Componente que gestiona los logs y ofrece la posibilidad bien de almacenarlos en disco o de comprimirlos en un archivo zip y subirlos al sistema de ficheros, en este caso HDFS.

- ***ContainerExecutor:*** Componente encargado de interactuar con el sistema operativo subyacente para alojar los archivos y directorios en las rutas correctas y controlar los procesos referentes a los contenedores.

- ***NodeHealthCheckerService:*** Componente encargado de chequear la salud de los nodos mediante la ejecución un script. Además, chequea la salud del disco creando archivos temporales cada cierto tiempo. En el momento en el que se produce un cambio en el estado del nodo, el *NodeStatusUpdater* es avisado y este le transmitirá la información al RM.

- ***Seguridad:*** Dos subcomponentes forman el servicio de seguridad del NodeManager.

  - ***ApplicationACLsManager:*** Componente encargado de mantener las ACLs para auntenticar y autorizar peticiones. Estas peticiones pueden venir de diferentes fuentes, ya sea la interfaz web o alguna API controlada por un usuario.
  - ***ContainerTokenSecretManager:*** Verifica peticiones entrantes para garantizar que están verdaderamente autorizadas por el RM.

- ***Servidor Web:*** Componente que ofrece información sobre las aplicaciones, contenedores corriendo en un determinado nodo en un momento de tiempo, salud del nodo y los logs producidos por los contenedores.

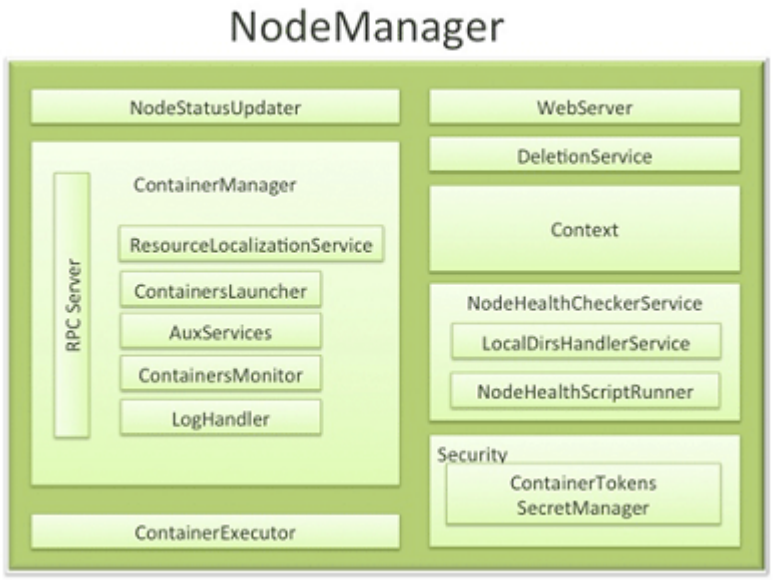


FIGURA 4.15: Diferentes componentes que forman el Node Manager del gestor de recursos YARN.

DataNode de HDFS

Los DataNodes [23] son los verdaderos trabajadores del sistema de ficheros. Son los encargados de guardar bloques cuando se les manda esa tarea y de reportar al NameNode periódicamente la lista de bloques que guardan.

Cada una de estas réplicas que guardan los DataNodes es representada por dos archivos en el sistema de ficheros local de la máquina que ejecuta el DataNode. El primero de ellos guarda los datos actuales del bloque y por tanto su tamaño coincide con el del bloque. Por otra parte, el segundo archivo guarda metadatos del bloque, incluyendo un checksum y un timestamp.

Una vez se levanta un DataNode, se realiza un handshake con el NameNode para comprobar el namespace ID del DataNode y la versión del software. Si alguno de ellos no correspondiera con el valor esperado, el NameNode lo terminaría. Para garantizar la integridad del cluster, el namespace ID es único y persistente, por tanto sólo se otorga una vez a un DataNode en el momento en el que se une al cluster.

Una vez se ha realizado el handshake, el DataNode queda registrado en el NameNode con un storage ID único que guarda el DataNode. Este ID es persistente e identifica el DataNode a pesar de que este cambie de

dirección IP o de puerto. Al igual que el namespace ID, este identificador es asignado en el momento en que el DataNode se une al cluster y a partir de ese momento no cambia. El DataNode envía información al NameNode sobre las réplicas que posee mediante los llamados block reports. En ellos se envía el block id, el generation stamp y el tamaño del bloque en el host. Estos mensajes se envían la primera vez que un DataNode se registra y posteriormente se envían periódicamente.

Para informar al NameNode del estado de salud, cada DataNode envía cada 3 segundos un heartbeat. Como se ha visto en la sección dedicada al NameNode, si este no recibiera un heartbeat de un DataNode en un intervalo de 10 minutos, se terminaría su ejecución. Si esto llega a suceder, se redistribuirían las réplicas que ese DataNode guardase. Es importante tener en cuenta que el NameNode no se comunica directamente con el DataNode, sino que responde a los heartbeats para realizar las diferentes acciones, ya sea replicar bloques de otro DataNode, eliminar réplicas o bloques, volver a registrar o eliminar un nodo o enviar un block report.

Una vez descrito el principal funcionamiento de un DataNode dentro un cluster Hadoop, se observa que en nuestro cluster se tiene un DataNode en cada servidor, por tanto se tiene dos DataNodes. Dentro de la interfaz web del NameNode se puede obtener cierta información sobre los DataNodes, contando además con el explorador de archivos.

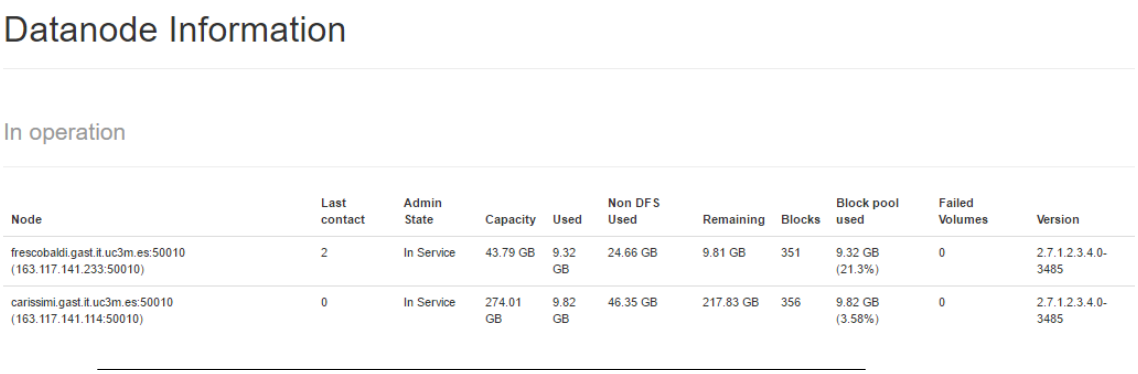


FIGURA 4.16: Información sobre los DataNodes que ofrece la interfaz gráfica web del NameNode, ubicada en el puerto 50070.

ZooKeeper Server

La principal misión de ZooKeeper es permitir la coordinación entre sistemas distribuidos. A pesar de que la idea de tener varios procesos ejecutándose en un mismo servidor no es tan diferente de la ejecución de servicios en un cluster, la principal diferencia es que típicamente en un cluster no se encuentran recursos compartidos por varias máquinas, sino que lo que comparten las máquinas es la red.

La opción más sensata para garantizar la coordinación dentro del cluster parece ser confiar en un componente que proporcione almacenamiento compartido con algún sistema que garantice prioridades en el acceso: ZooKeeper [26].

ZooKeeper almacena información en árboles jerárquicos donde cada nodo es llamado znode. La figura 4.17 muestra un ejemplo de la organización de los znodes en ZooKeeper.

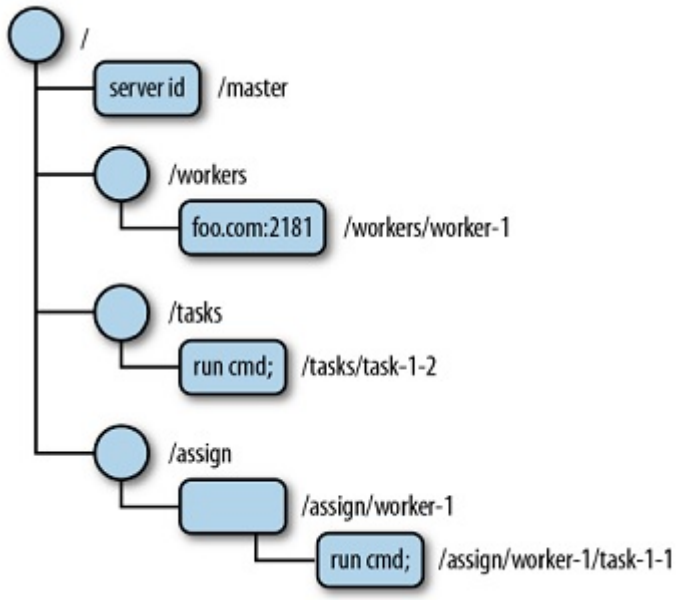


FIGURA 4.17: Organización que realiza ZooKeeper de la información, diferenciando entre trabajadores, tareas y la función de asignación.

- El znode */workers* representa todos los trabajadores que están disponibles en el cluster. Es el padre de todos aquellos nodos, y en este ejemplo, el nodo foo.com:2181 está disponible. Si algún nodo presente en el árbol de /workers dejara de estar disponible, sería eliminado.
- El znode */task* es el nodo padre de todos aquellos nodos que representan tareas que se encuentran esperando a ser adjudicadas a un trabajador.
- El znode */assign* es el nodo padre de todos aquellos nodos que representan la acción de una tarea siendo asignada a un trabajador.

ZooKeeper presenta una API para gestionar los znodes, dentro de las acciones que se pueden realizar encontramos:

- ***create /path data:*** Crea un znode en la ruta designada por path con la información contenida en data.
- ***delete /path:*** Elimina el znode en la ruta path.
- ***exists /path:*** Comprueba si el znode de la ruta path existe.
- ***setData /path data:*** Modifica la información del znode en path con la información data.
- ***getData /path:*** Obtiene la información del znode en la ruta path.

- ***getChildren:*** Obtiene una lista con los znodes hijos del znode especificado en path.

La interfaz web que presenta Ambari nos muestra información sobre el servidor ZooKeeper ejecutándose y los clientes de ZooKeeper.

FIGURA 4.18: Información que proporciona Ambari sobre el servidor ZooKeeper ejecutándose así como el número de clientes instalados.

Enhanced Configurations de Ambari

Ambari proporciona una interfaz web para la configuración de cada uno de los servicios instalados sin tener que modificar el código de los ficheros de configuración presentes en cada servidor. Se facilita de este modo la configuración de los servicios al contar con sliders, valores máximos y mínimos establecidos por otros parámetros, conversión de unidades automática y recomendación de valores [27].

Un ejemplo de la interfaz web que proporciona Ambari para la configuración de YARN se puede ver en la figura 4.19.

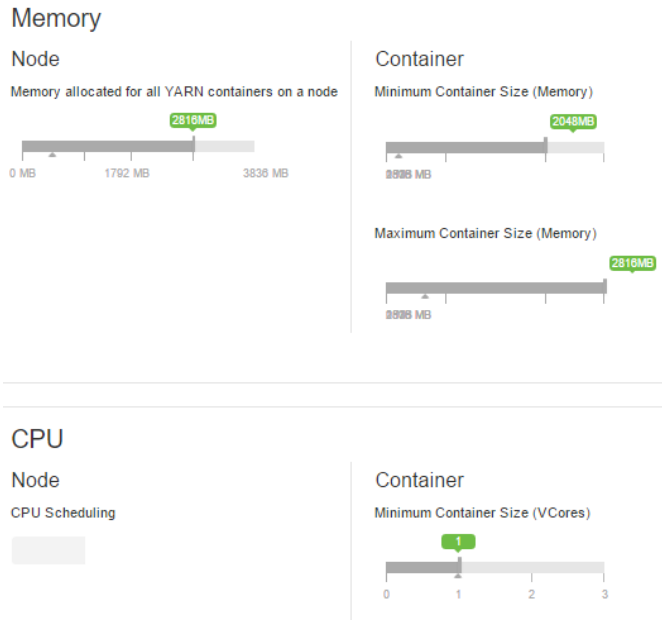


FIGURA 4.19: Diferentes elementos visuales que ofrece Ambari para modificar la configuración de YARN sin tener que modificar los archivos de configuración del servidor.

El principal potencial de esta herramienta es el control de versiones de configuración, es decir, cada cambio realizado en la configuración de cada servicio queda registrado bajo una versión con comentarios opcionales sobre las modificaciones realizadas. Del mismo modo, las versiones anteriores quedan registradas por si se quisiera volver a una configuración anterior.

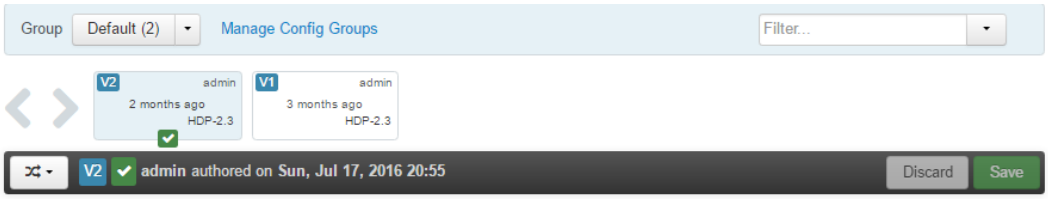


FIGURA 4.20: Diferentes versiones de configuración de YARN a lo largo del tiempo.

4.2. Segundo Prototipo

Para este segundo prototipo se cuenta con 5 máquinas ubicadas en el laboratorio 4.1B01. Los FQDN de los servidores son los siguientes:

- ***Doc026.lab.it.uc3m.es***
- ***Doc027.lab.it.uc3m.es***
- ***Doc028.lab.it.uc3m.es***
- ***Doc029.lab.it.uc3m.es***
- ***Doc030.lab.it.uc3m.es***

La planificación que se ha sugerido para el cluster es la siguiente. El master sería el ordenador *Doc027*, mientras que los demás ejercerían de nodos esclavos. Dado que elegir que el *Doc027* funcione a su vez como nodo esclavo, se ha decidido que también ejerza de agente de Ambari, por lo que se cuenta con un master y 5 agentes esclavos.

Durante el proceso de instalación y posterior despliegue de los servicios requeridos por Ambari, se han encontrado multitud de problemas que se relatarán más adelante en el capítulo 6 y que han impedido que dicha instalación se lleve a cabo. Por tanto, no se ha podido completar este segundo prototipo.

4.3. Arquitectura

4.3.1. Instalación

La versión del servidor de Apache Ambari elegida para la instalación es la 2.2.0. Esta versión está disponible para los siguientes sistemas operativos:

- ***(Redhat/CentoOS/Oracle) 6.***
- ***(Redhat/CentoOS/Oracle) 7.***
- ***SUSE 11.***
- ***Ubuntu 12.***
- ***Debian 7.***

La instalación se seguirá sobre Debian 7 como sistema operativo. Si se quisiera instalar en otro únicamente habría que cambiar el URL del repositorio y seguir las instrucciones de instalación indicadas en la documentación oficial. La instalación se deberá realizar sobre máquinas con procesadores de 64 bits exclusivamente; sobre máquinas de 32 bits no será posible instalar el software.

En primer lugar se actualiza el directorio sources.list.d ubicado en la carpeta apt. Desde dicho directorio se procede a descargar el repositorio de la forma descrita en la siguiente figura.

```
jmgallego@carissimi:/etc/apt/sources.list.d$ cd /etc/apt/sources.list.d/
jmgallego@carissimi:/etc/apt/sources.list.d$ sudo wget http://public-repo-1.hort
onworks.com/ambari/debian7/2.x/updates/2.2.0.0/ambari.list
```

FIGURA 4.21: Descarga del repositorio ambari.list.

Importamos las claves gpg mediante el siguiente comando:

```
jmgallego@carissimi:/etc/apt/sources.list.d$ sudo apt-key adv --recv-keys --keys
erver keyserver.ubuntu.com B9733A7A07513CAD
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --secret
-keyring /tmp/tmp.w7bNCnMxoN --trustdb-name /etc/apt//trustdb.gpg --keyring /etc
/apt/trusted.gpg --primary-keyring /etc/apt/trusted.gpg --keyring /etc/apt/trust
ed.gpg.d//debian-archive-jessie-automatic.gpg --keyring /etc/apt/trusted.gpg.d//
debian-archive-jessie-security-automatic.gpg --keyring /etc/apt/trusted.gpg.d//d
ebian-archive-jessie-stable.gpg --keyring /etc/apt/trusted.gpg.d//debian-archive
-squeeze-automatic.gpg --keyring /etc/apt/trusted.gpg.d//debian-archive-squeeze-
stable.gpg --keyring /etc/apt/trusted.gpg.d//debian-archive-wheezy-automatic.gpg
 --keyring /etc/apt/trusted.gpg.d//debian-archive-wheezy-stable.gpg --recv-keys
--keyserver keyserver.ubuntu.com B9733A7A07513CAD
gpg: solicitando clave 07513CAD de hkp servidor keyserver.ubuntu.com
gpg: clave 07513CAD: "Jenkins (HDP Builds) <jenkin@hortonworks.com>" sin cambios
gpg: Cantidad total procesada: 1
```

FIGURA 4.22: Importación clave gpg.

Y finalizamos la instalación ejecutando haciendo uso de apt-get. Este programa busca en su base de datos la versión más reciente de paquete y lo descarga del repositorio especificado previamente en sources.list.d:

apt-get update
apt-set install ambari-server

Una vez instalado el servidor de Ambari, se procederá a la configuración del servidor mediante el comando: *ambari-server setup*.

Durante la configuración del setup, se podrá elegir el JDK a utilizar, pudiéndose descargar directamente con las opciones 1 y 2 o estableciendo como JDK el ya instalado en la máquina. En este caso, se elegirá la opción 3, se define el path de Java ya instalado en la máquina como el path de JAVA_HOME.

Una vez configurado el servidor, se podrá levantar y seguir con el proceso de instalación del cluster a través de la interfaz web que Ambari proporciona. Esta Web UI se encuentra disponible una vez levantado el servidor en *http://<ambari-server-host>:8080*. Dependiendo de las características de las máquinas, el proceso de arrancar del servidor puede tomar algún tiempo. Si este es el caso, no se podrá realizar la conexión al puerto 8080; simplemente habría que esperar a que termine de arrancar el servidor para poder conectarse. Por defecto, se crea el usuario administrador con usuario *admin* y contraseña *admin*; estos credenciales se pueden modificar más adelante, así como crear otros administradores del cluster o grupos con determinados privilegios.

```
jmgallego@carissimi:/etc/apt/sources.list.d$ sudo ambari-server start
Using python  /usr/bin/python2
Starting ambari-server
Ambari Server running with administrator privileges.
About to start PostgreSQL
Organizing resource files at /var/lib/ambari-server/resources...
WARNING: setpgid(2394, 0) failed - [Errno 13] Permission denied
Server PID at: /var/run/ambari-server/ambari-server.pid
Server out at: /var/log/ambari-server/ambari-server.out
Server log at: /var/log/ambari-server/ambari-server.log
Waiting for server start.....................
Ambari Server 'start' completed successfully.
```

FIGURA 4.23: Arranque del servidor Ambari.

La interfaz de entrada nos permite crear un cluster, administrar permisos de usuarios y grupos y lanzar views de servicios personalizados.

La opción de "Launch Install Wizard"lleva directamente al proceso de creación del cluster. Se podrá elegir el nombre del cluster en primer lugar. Una vez elegido el nombre del cluster y la stack de Hadoop que se utilizará se procede a la incorporación de las diferentes máquinas al cluster. Durante esta instalación se elige la stack de Hadoop 2.3. El siguiente paso a tomar es el registro de los hosts que formarán parte del cluster, ya sea el master como los esclavos, se definen todos. Más adelante, terminada la creación del cluster, se podrán añadir o eliminar máquinas del cluster. Para identificar a los hosts se deberá introducir el nombre del dominio (FQDN) de la máquina.

El registro se puede terminar de dos formas distintas: bien añadiendo la clave SSH privada de cada host para un registro automático; o bien mediante la instalación manual de los agentes en aquellos hosts que formarán

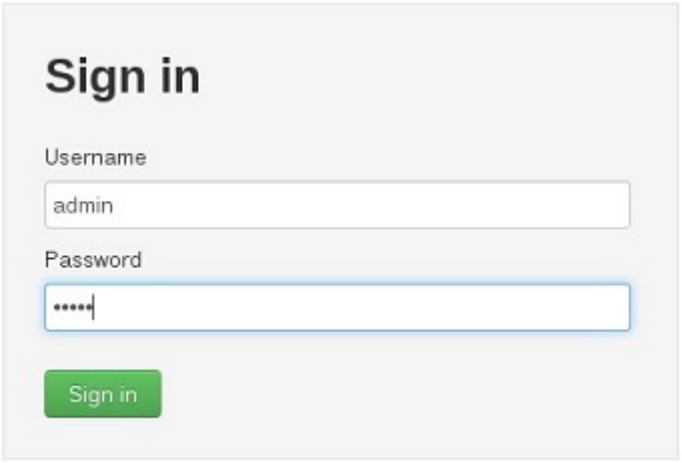


FIGURA 4.24: Entrada a la interfaz web que proporciona el servidor.

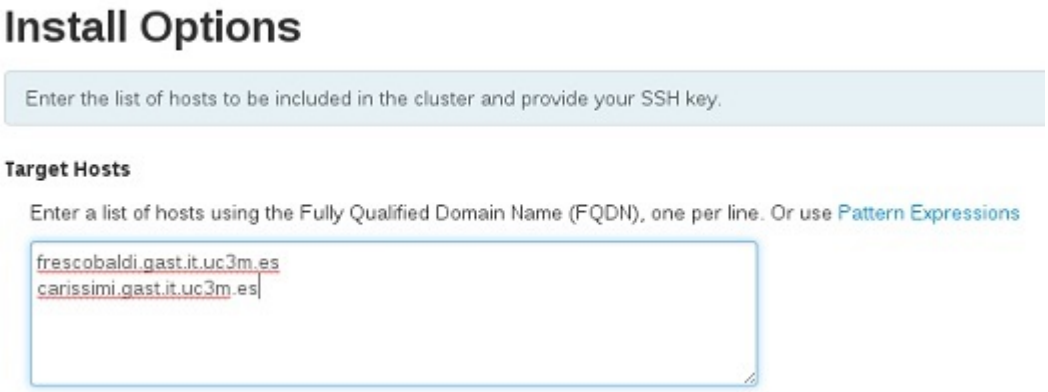


FIGURA 4.25: Incorporación de hosts al cluster mediante los FQDNs.

parte del cluster y posterior registro. Se utilizará el registro manual de los hosts. Para ello, una vez pasado este punto, debe tenerse en cuenta que los agentes Ambari deben estar instalados y ejecutándose en cada host.

Para instalar los agentes Ambari primero descargamos el ambari.repo y comprobamos que los paquetes se han descargado correctamente. Estos son los comandos necesarios para instalar los agentes correspondientes a Ambari 2.2.0 en Ubuntu 14:

wget -nv http://public-repo-1.hortonworks.com/ambari/ubuntu14/2.x/updates/2.2.0.0/ambari.list -O /etc/apt/sources.list.d/ambari.list
apt-key adv –recv-keys –keyserver keyserver.ubuntu.com B9733A7A07513CAD
apt-get update
apt-cache showpkg ambari-server
apt-cache showpkg ambari-agent
apt-cache showpkg ambari-metrics-assembly

Por último, se termina la instalación con el comando *apt-get install ambari-agent* y se configura mediante el archivo ambari-agent.ini para que quede de la siguiente manera. La ruta de acceso al fichero es */etc/ambari-agent/conf/ambari-agent.ini*.

```
[server]
hostname=carissimi.gast.it.uc3m.es
url_port=8440
secured_url_port=8441

[agent]
prefix=/var/lib/ambari-agent/data
;loglevel=(DEBUG/INFO)
loglevel=INFO
```

FIGURA 4.26: Fichero de configuración del agente de Ambari donde se le dice cuál es el master y los puertos a utilizar.

Llegado este punto, ya se puede levantar el agente de Ambari: *ambari-agent start*.

Retomando la instalación del servidor, habiéndose asegurado que los agentes están instalados y ejecutándose de la forma descrita, se procede a la confirmación de los hosts.

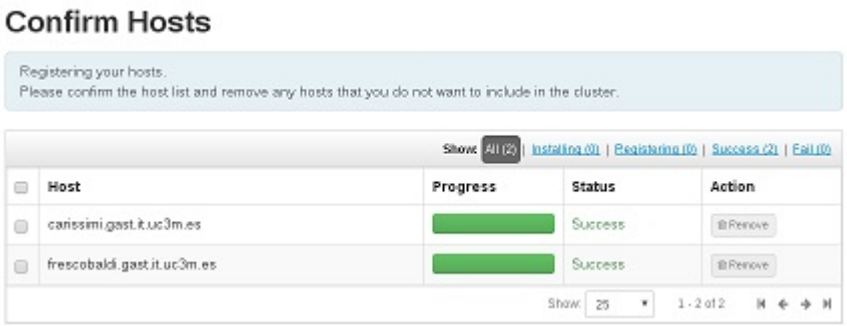


FIGURA 4.27: Confirmación satisfactoria de hosts al cluster.

A continuación se pueden elegir los servicios que instalar en el cluster. Parte de ellos se han estudiado en este trabajo y se elegirán: HDFS, YARN, MapReduce, ZooKeeper, Ambari Metrics y Spark; siendo posible más adelante instalar servicios adicionales si se quisiera. Dependiendo de las características de las máquinas, su rendimiento se verá afectado por el número de servicios que se seleccionen, ya que serán servicios que estarán ejecutándose de manera continua en el cluster. Por tanto, pudiéndose añadir servicios en un punto más avanzado, se eligen esos servicios para la instalación.

En la siguiente pantalla se pueden elegir los masters para los servicios elegidos a través de los diferentes hosts, pudiendo elegir el más adecuado para cada servicio según las características de la máquina: capacidad de cómputo, número de núcleos, memoria RAM... La estrategia que se seguirá será situar aquellos servicios que requieran más recursos, como los servidores, en los ordenadores más potentes. Aquellos servicios que sirven de

back up a otros, como por ejemplo el SNameNode que recordamos sirve para guardar puntos de control y configuraciones del NameNode, se recomienda instalar en máquinas distintas, por el simple motivo de posibilidad de recuperación en caso de fallo. Como se ha mencionado, el cluster tiene que presentar tolerancia a fallos.

Se elegirán también los servicios a instalar en cada host y la configuración de cada uno de ellos. Esta configuración que se puede establecer en la interfaz gráfica queda guardada en los hosts en la ruta */usr/hdp/2.3.4.0-3485*. Este último directorio hace referencia a la versión de la stack de Hadoop instalada, si se hubiera elegido otra versión esta parte final del path habría sido diferente. También se encuentran los archivos de configuración en el otro directorio de aparece dentro de */usr/hdp*, llamado *current*.

Capítulo 5

Pruebas

Una vez terminada la instalación del cluster se procede a la realización de las diversas pruebas con la finalidad de apreciar las diferencias en las posibles configuraciones de los componentes. Las pruebas se han estructurado para evaluar el rendimiento en diferentes aspectos del cluster y se realizarán tanto de manera local en una sola máquina como en el cluster.

La página principal de Ambari ofrece varias métricas en forma de gráficas que serán muy útiles para apreciar los cambios en tráfico de red, uso de CPU, gestión de memoria por parte de YARN, espacio ocupado del HDFS, entre otros.

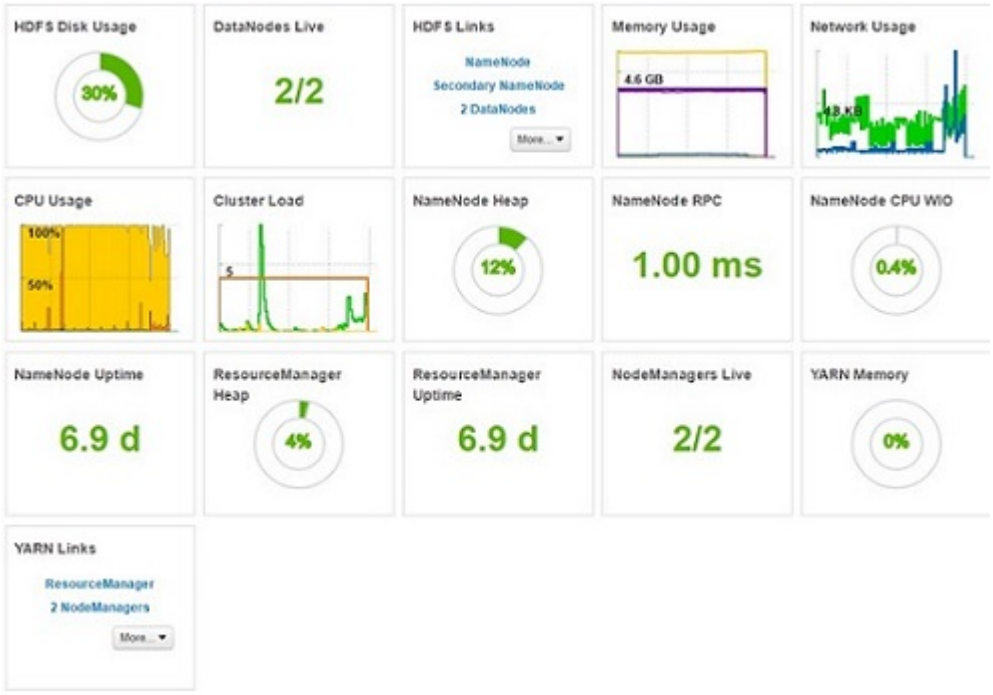


FIGURA 5.1: Diferentes métricas que ofrece Ambari sobre el estado del cluster: uso de disco, uso de la red, uso de CPU, carga del cluster...

La pruebas que se van a realizar se ejecutarán sobre Spark. Como ya se ha mencionado, Spark se ejecuta en este cluster sobre la gestión de recursos de YARN, por tanto se hará uso de ambos componentes. En primer lugar se realizará una prueba básica incluida en los ejemplos de Spark para comprobar que todo funciona correctamente, se trata de la prueba SparkPi.

Seguidamente se realizará un WordCount sobre ficheros de diferente tamaño cambiando además diferentes parámetros.

La ejecución de todas las pruebas se realizará siguiendo el mismo procedimiento, este es, mediante el *Spark-submit* que ofrece el cliente de Spark. La ruta al ejecutable *Spark-submit* es la siguiente:

/usr/hdp/current/spark-client/bin/spark-submit

Dentro de los diferentes parámetros que se le pueden pasar al *Spark-submit*, se le pasará la clase a ser ejecutada, el master de la ejecución, que en nuestras pruebas será YARN y el archivo .jar donde se encuentra la clase a ejecutar. Opcionalmente según lo requiera el código se introducirán más argumentos, como en el caso de los WordCount.

5.1. Estructura de las pruebas

Las pruebas que se realizarán podrán separarse entre las ejecutadas de manera local en el master *carissimi* y las ejecutadas de manera distribuida en *carissimi* y *frescobaldi*. Seguirán la estructura mostrada en la tabla 5.1.

Con el objetivo de que las pruebas sean lo más objetivas posible serán ejecutadas de manera aislada en las máquinas, intentando que sea lo único ejecutándose en el momento. De esta manera se pueden comparar los resultados de manera más legítima y se pueden extraer conclusiones aceptables.

Ejecución	Primera prueba	Segunda prueba
Local en *carissimi*	WordCount 500MB	WordCount 6GB
Distribuida en *carissimi* y *frescobaldi*	WordCount 500MB	WordCount 6GB

CUADRO 5.1: Organización de las pruebas.

5.2. SparkPi

Siguiendo las indicaciones previas para la ejecución del programa, se crea un archivo .sh que se llamará SparkPi.sh con la siguiente información:

/usr/hdp/current/spark-client/bin/spark-submit –class org.apache.spark.examples.SparkPi –master yarn /usr/hdp/current/spark-client/lib/spark-examples-1.5.2.2.3.4.0-3485-hadoop2.7.1.2.3.4.0-3485.jar

Para ejecutarlo, se escribe desde el terminal:

sh SparkPi.sh

La información que revela el ResourceManager sobre la aplicación es la siguiente:

User:	jmgallego
Name:	Spark Pi
Application Type:	SPARK
Application Tags:	
YarnApplicationState:	FINISHED
Queue:	default
FinalStatus Reported by AM:	SUCCEEDED
Started:	sáb jul 30 17:03:58 +0200 2016
Elapsed:	9mins, 47sec
Tracking URL:	History
Log Aggregation Status	NOT_START
Diagnostics:	null

FIGURA 5.2: Detalle de la aplicación SparkPi que ofrece el ResourceManager: usuario, nombre, estado final, tiempo utilizado...

El programa muestra por consola junto a una lista de mensajes de información la siguiente aproximación de pi: **Pi is roughly 3.14146**.

La información que se va a analizar en primer lugar es el tráfico de red y la CPU. Se observan ambas en las figuras 5.4 y 5.5. El tráfico de red aumenta como era previsible en el momento de lanzar la tarea. Esto se debe al carácter de funcionamiento de MapReduce y el paradigma de distribuir los trabajos y no los datos. En este programa no hay datos que distribuir para trabajar sobre ellos, sino que se distribuye el archivo .jar a los nodos que vayan a ejecutar el trabajo. Según la trama de información que arroja el terminal, vemos como el archivo se envía al hdfs en la figura 5.3. En referencia al al uso de la CPU, no se aprecia que este suba considerablemente, por tanto en otras pruebas se intentará hacer un uso más intensivo de la CPU.

```
16/07/30 17:00:28 INFO Client: Uploading resource file:/usr/hdp/2.3.4.0-3485/spa
rk/lib/spark-assembly-1.5.2.2.3.4.0-3485-hadoop2.7.1.2.3.4.0-3485.jar -> hdfs://
carissimi.gast.it.uc3m.es:8020/user/jmgallego/.sparkStaging/application_14698712
36192_0003/spark-assembly-1.5.2.2.3.4.0-3485-hadoop2.7.1.2.3.4.0-3485.jar
16/07/30 17:03:50 INFO Client: Uploading resource file:/tmp/spark-3a6e0903-3448-
459b-acc4-aeb43c038e23/__spark_conf__4347194292275215882.zip -> hdfs://carissimi
.gast.it.uc3m.es:8020/user/jmgallego/.sparkStaging/application_1469871236192_000
3/__spark_conf__4347194292275215882.zip
```

FIGURA 5.3: Subida del programa comprimido en .jar al hdfs a los nodos trabajadores.

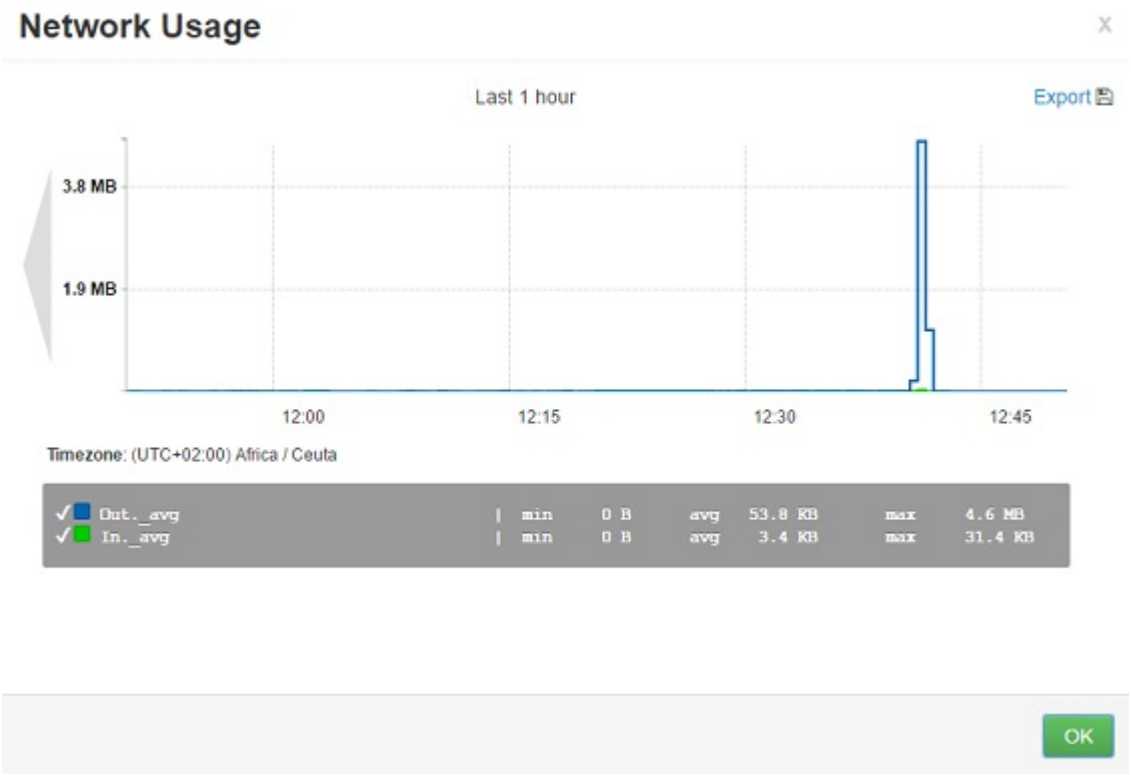


FIGURA 5.4: Aumento del tráfico out en el master debido a la distribución del trabajo.

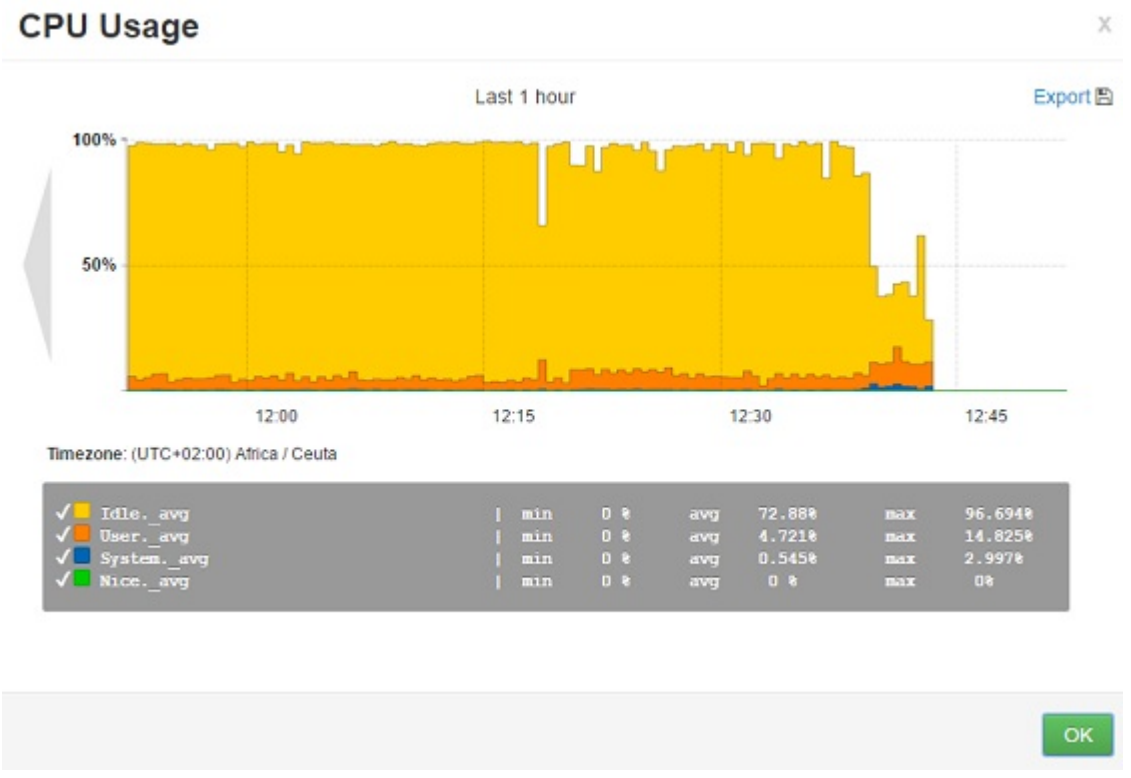


FIGURA 5.5: Uso de la CPU durante la ejecución de la prueba SparkPi.

5.3. WordCount

Tras comprobar que la instalación es correcta y el cluster es capaz de ejecutar aplicaciones como el SparkPi, se procede al estudio de otras tareas más versátiles para comprobar el comportamiento del cluster ante diferentes exigencias. Se ha programado un contador de palabras que funciona utilizando la capacidad de cómputo de Spark basándose en el paradigma clave valor, al igual que hace MapReduce. El código recibe como argumentos (se pasarían como argumentos al *Spark-submit*) el archivo de texto de entrada y el directorio donde se colocará el resultado. Estas dos rutas pueden ser tanto locales como rutas al HDFS. La forma de referenciar un archivo dentro de hdfs es la siguiente:

hdfs://<servidor donde se encuentra el NameNode>:8020/<ruta al archivo dentro de HDFS>

5.3.1. Subida de archivos al HDFS

Con el propósito de disponer de diferentes pruebas, se han subido al HDFS varios archivos de texto de diferentes longitudes. Para subir archivos al hdfs se utiliza la siguiente línea:

hdfs dfs -put <ruta local del archivo><ruta deseada en HDFS>

Con el fin de ver el impacto sobre el tráfico de red que tiene la configuración del HDFS, se va a subir un archivo de texto plano de 5,85 GB con el que después de realizarán diferentes pruebas. El actual nivel de replicación configurado en el HDFS es el valor predeterminado por defecto, 3 con unos bloques de 128 MB. Al contar con dos nodos en el cluster y un nivel de replicación de 3, todos los bloques estarán disponibles en los dos nodos, tanto en *carissimi* como en *frescobaldi*. En las figuras 5.7 y 5.8 vemos el aumento del tráfico de red y del uso de CPU.

Tras la subida de los diferentes archivos al sistema de ficheros distribuidos, se cuenta con 5 archivos de texto que van desde los 6,19 MB hasta los 5,85 GB de tamaño.

-rw-r--r--	jmgallego	jmgallego	6.19 MB	2/6/2016 16:02:28	3	128 MB	big.txt
-rw-r--r--	jmgallego	jmgallego	170.46 MB	27/7/2016 12:28:50	3	128 MB	libro.txt
-rw-r--r--	jmgallego	jmgallego	495.5 MB	1/8/2016 11:02:15	1	128 MB	libro3.txt
-rw-r--r--	jmgallego	jmgallego	991 MB	27/7/2016 23:58:49	3	128 MB	libro4.txt
-rw-r--r--	jmgallego	jmgallego	5.85 GB	30/7/2016 18:22:37	3	128 MB	libro5.txt

FIGURA 5.6: Detalles de los 5 archivos disponibles para el WordCount que ofrece el explorador de archivos del Name Node.

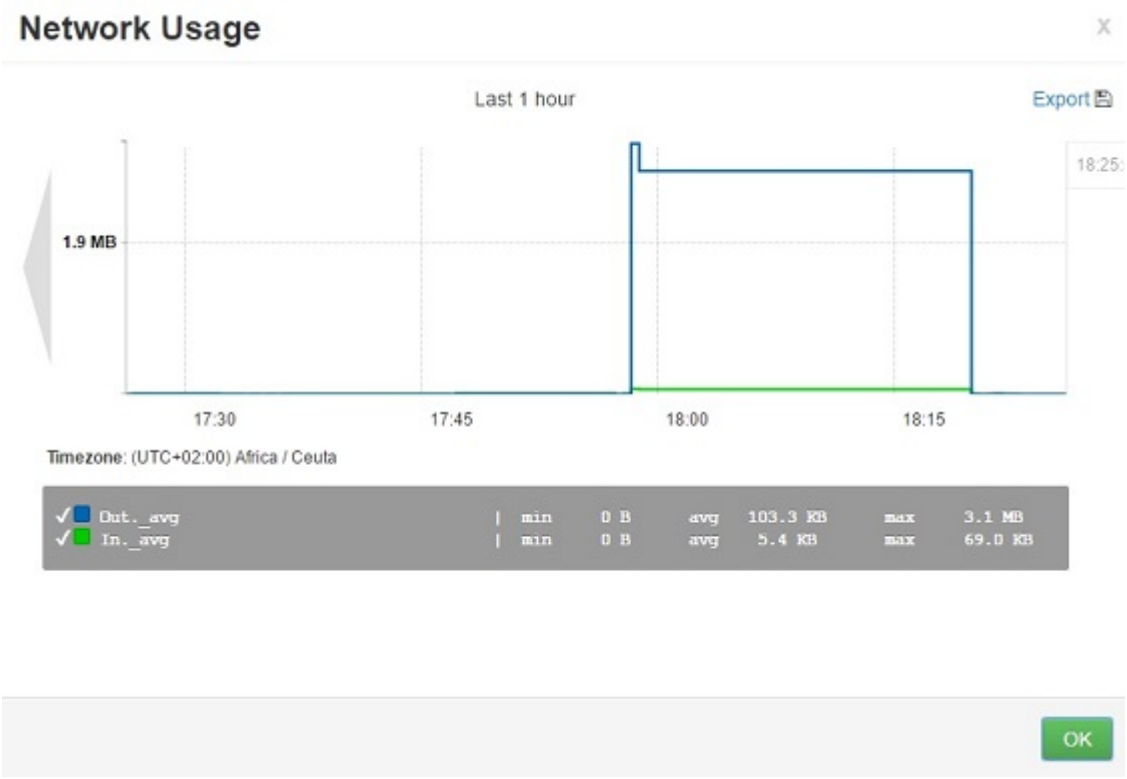


FIGURA 5.7: Incremento del tráfico de red al subir un archivo de 5,85 GB a HDFS.

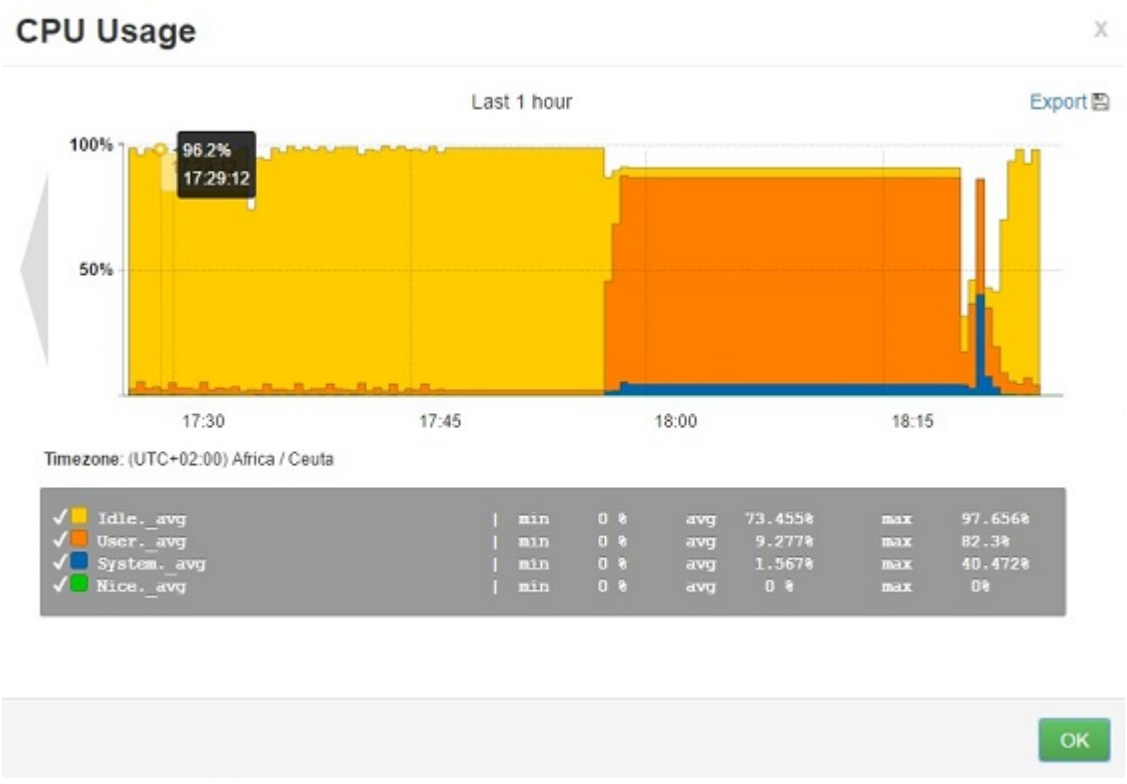


FIGURA 5.8: Incremento del uso de la CPU al subir un archivo de 5,85 GB a HDFS.

5.3.2. WordCount distribuido con 500MB

En esta prueba se realizará el WordCount con un archivo de 495,5 MB. El tiempo empleado es de 11 minutos 42 segundos. Analizando el uso de la red vemos que no se aprecia un gran incremento del tráfico. Esto se debe principalmente a dos motivos:

- Al contar con únicamente dos nodos y usar un valor de replicación de 3, todos los datos se encuentran distribuidos en ambos nodos. Esto quiere decir que cada nodo dispone de toda la información necesaria para la tarea, no existe la posibilidad de que se requiera información extra que posea otro nodo del cluster y de esta forma utilizar la red para distribuir datos.
- El único uso que se realiza de la red es para distribuir el archivo comprimido con la tarea. Debido a esto el tráfico se incrementa al principio de la tarea mientras que el incremento de la CPU se observa a partir de ese pico en el tráfico.

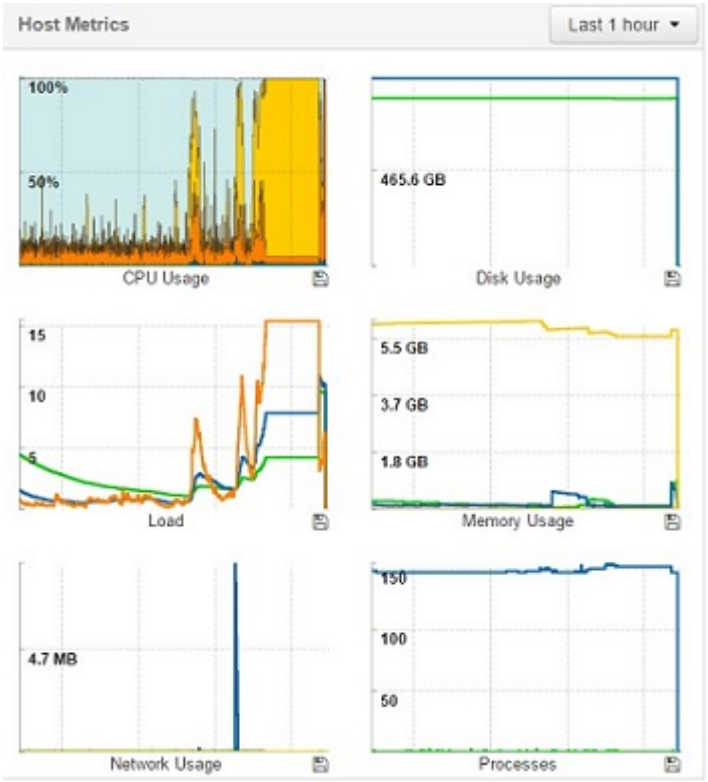


FIGURA 5.9: Diferentes métricas que muestra Ambari sobre las métricas al realizar el WordCount distribuido con un archivo de 500MB.

5.3.3. WordCount local con 500MB

Para intentar comprobar la diferencia entre contar con dos nodos o únicamente un nodo, se va a ejecutar la misma tarea, un WordCount de 500MB, de manera local. El script a ejecutar contendría las siguientes líneas:

/usr/hdp/2.3.4.0-3485/spark/bin/spark-submit –class pruebaSpark.prueba –master local /home/jmgallego/Escritorio/pruebaSpark2/target/pruebaSpark-0.0.1-SNAPSHOT.jar hdfs://163.117.141.114:8020/user/jmgallego/libro3.txt hdfs://163.117.141.114:8020/user/jmgallego/pruebaWordCountLibro3Local

Dado que esta aplicación se ejecuta en el local y ya no utiliza YARN como responsable de la gestión de recursos, sino que se ejecuta en modo standalone, para su monitorización se visita el Spark History Server ya que es el encargado de gestionar los recursos. Por tanto, se visita *https://carissimi.gast.it.uc3m.es:4040*.

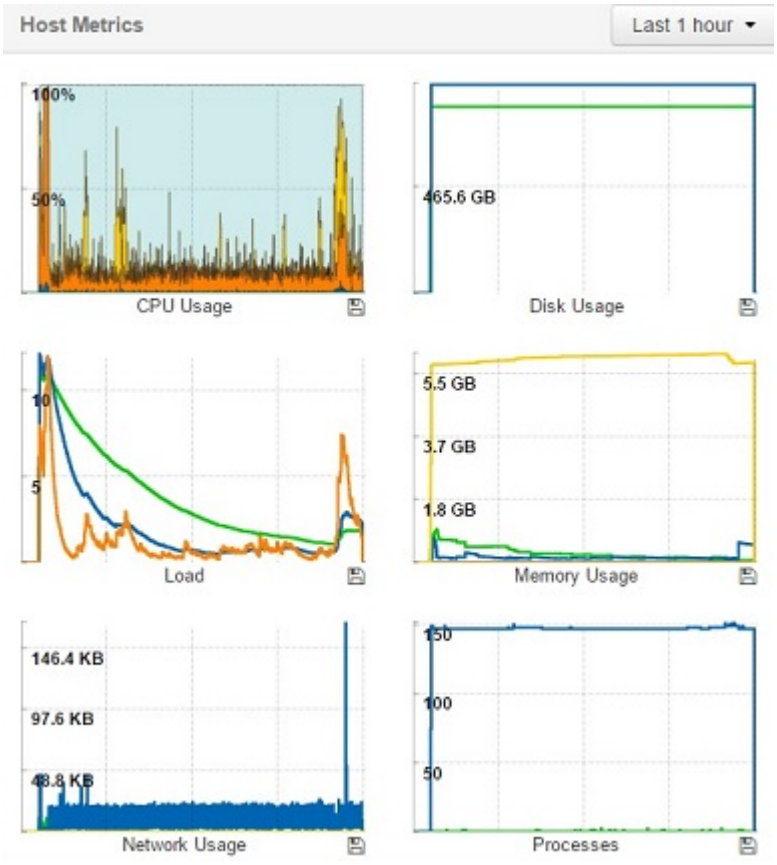


FIGURA 5.10: Diferentes métricas que muestra Ambari sobre las métricas al realizar el WordCount local con un archivo de 500MB.

La tarea se ha completado con éxito en 2 minutos 16 segundos, bastante menos tiempo que la empleada para realizar la misma tarea manera distribuida. Por otra parte, y a pesar de que el uso de red en esta prueba local es menor que en la prueba distribuida, el uso de la CPU es mayor, es decir, se requiere mayor rendimiento del servidor.

5.3.4. WordCount distribuido con 6GB

Esta prueba intentará ser la más representativa del comportamiento de un cluster ante una tarea de Big Data real. Se realizará el WordCount con un archivo de texto plano de 5,85 GB con la configuración utilizada previamente.

La tarea se ha realizado en 22 minutos 55 segundos con un estado final de "SUCCEEDED". Las métricas de *carissimi* son las mostradas en la figura 5.11. El aumento de tráfico de red correspondiente a esta tarea es el que se observa a mitad de la gráfica. Se puede ver que es prácticamente igual que el tráfico para la prueba anterior de 500MB, por tanto nos aseguramos que el tráfico de red no se ve afectado por la cantidad de texto a procesar, debido a la distribución de la tarea.

Por otra parte, el uso de la CPU no sufre un gran aumento y debido a esto es por lo que se necesita más tiempo para completar la tarea. Para este tipo de tareas, parece que no es suficiente un cluster de únicamente dos nodos y para observar el rendimiento real que puede ofrecer Spark en tareas distribuidas necesitaríamos más ejecutores.

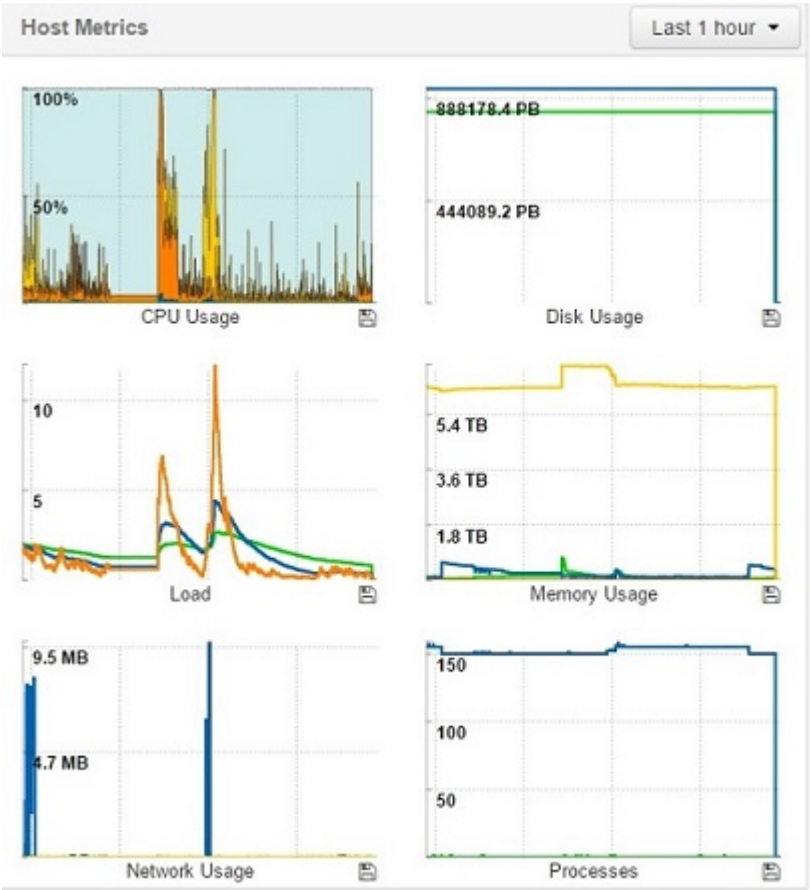


FIGURA 5.11: Diferentes métricas que muestra Ambari sobre las métricas al realizar el WordCount distribuido con un archivo de 6GB.

5.3.5. WordCount local con 6GB

A continuación se va a ejecutar la misma prueba de manera local. Se ejecuta el siguiente script:

/usr/hdp/2.3.4.0-3485/spark/bin/spark-submit –class pruebaSpark.prueba –master local /home/jmgallego/Escritorio/pruebaSpark2/target/pruebaSpark-0.0.1-SNAPSHOT.jar hdfs://163.117.141.114:8020/user/jmgallego/libro5.txt hdfs://163.117.141.114:8020/user/jmgallego/pruebaWordCountLibro5Local

La tarea se ha completado satisfactoriamente en 25 minutos y 35 segundos. Las métricas que ofrece Ambari de *carissimi*, que ha sido el nodo encargado de la ejecución, son las siguientes.

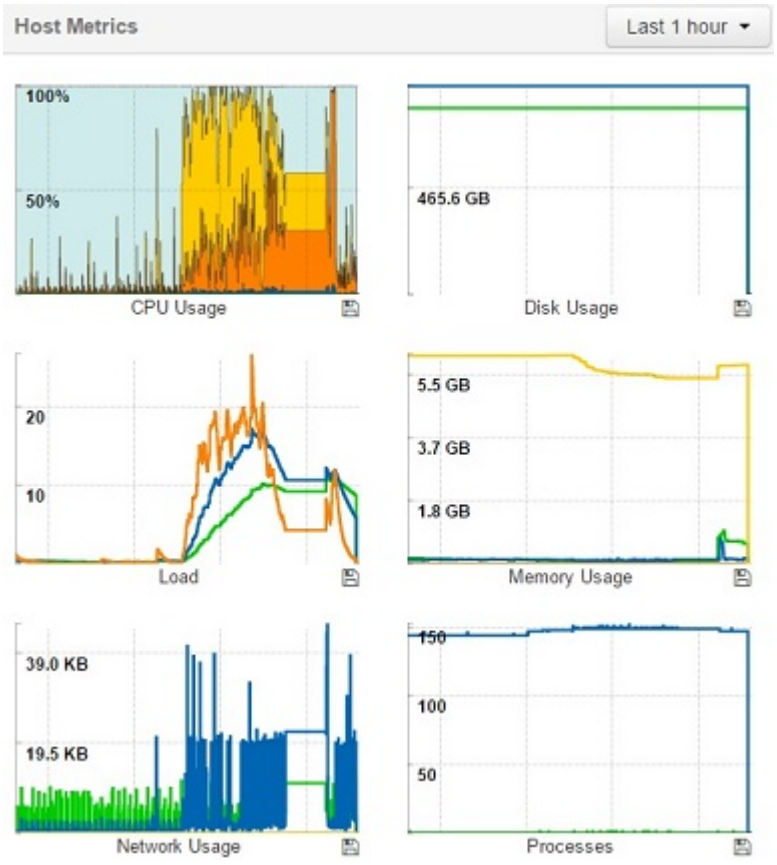


FIGURA 5.12: Diferentes métricas que muestra Ambari sobre las métricas al realizar el WordCount local con un archivo de 6GB.

Se observa que el tráfico de red se ha reducido con respecto a la prueba anterior, pero por el contrario, el uso de la CPU es más intenso al realizar la prueba de manera local.

Queda demostrado que para archivos grandes que pueden considerarse tareas reales de big data, se nota la diferencia entre una ejecución local y el uso de un cluster, aunque dicho cluster cuente únicamente con dos nodos para distribuir el trabajo. En el caso contrario, al realizar tareas que no sean muy exigentes, y por tanto no sean representativas de un problema de Big Data, se obtiene un mayor rendimiento ejecutándolas de manera local. Este resultado es muy ilustrador y pone de manifiesto la importancia de disponer de un cluster para comprobar la eficacia de una tarea de Big Data, puesto que el rendimiento del procesamiento distribuido de Spark aumenta a medida que aumenta la cantidad de trabajo a procesar.

Capítulo 6

Historia del Proyecto

Durante la realización de este proyecto han aparecido problemas que han afectado al desarrollo. En este capítulo se van a describir estos problemas y las soluciones que han permitido que el proyecto siguiera adelante.

6.1. Instalación en el laboratorio

Para la implementación del cluster en el laboratorio del departamento de ingeniería telemática se ha trabajado conjuntamente con los técnicos de dicho laboratorio. En la siguiente sección se detallan problemas que se han encontrado durante la totalidad de la realización del proyecto, sin embargo, en este apartado se va a detallar porqué no ha sido posible ese desarrollo en el laboratorio.

El primer inconveniente que hubo que solventar era la incompatibilidad del software de Ambari con los sistemas operativos de 32 bits. A pesar que los kernels de las máquinas de los laboratorios eran de 64 bits, los sistemas operativos montados eran de 32, así que hubo que cambiar las imágenes de todos los ordenadores recompilándolos con la versión indicada de Ambari. Además, el sistema operativo Debian soportado por Ambari es la versión 7. En los laboratorios se tiene instalada la versión Debian 8, por tanto, dada la imposibilidad de cambiar la versión como se hizo en la instalación en el prototipo 1, se tuvo que recompilar el software modificando las comprobaciones que Ambari realizaba sobre el sistema operativo.

Una vez se terminó de instalar correctamente el software y se montaron las imágenes en todas las máquinas de los laboratorios, se encontraron problemas a la hora de crear un cluster y registrar los nodos. Estos problemas fueron causados por temas de privilegios. Por motivos de seguridad, la configuración realizada por los técnicos establece que algunos ficheros solo se puedan abrir en modo lectura, y al intentar modificarlos Ambari, se lanzaba el error.

Los técnicos de laboratorio procedieron a modificar los permisos de aquellos archivos que Ambari requería, sin embargo, a la hora de crear servicios del cluster, como son Spark, YARN, HDFS, MapReduce, seguían apareciendo errores del mismo tipo.

Al añadir la limitación temporal con la que se contaba en el proyecto a todos estos problemas, se tiene que tener en cuenta que el trabajo realizado en el laboratorio se tuvo que realizar a partir del mes de agosto para no molestar a otros procesos docentes que pudieran estar realizándose durante el

curso, ha resultado imposible terminar la instalación del cluster en el laboratorio, tal y como se tenía previsto. No obstante, el proceso sigue en curso y falta muy poco para que se termine dicha instalación y futuro despliegue.

6.2. Problemas del proyecto

6.2.1. Incompatibilidad del Sistema Operativo

La primera toma de contacto con el software de Ambari se planteó sobre una máquina virtual con una imagen ligera, Light Ubuntu. Tras el montaje de la imagen y posterior instalación del sistema operativo, se descubrió que para ese sistema operativo no estaba disponible el software. Dado que para Debian sólo estaba disponible para la versión 7, se decidió optar por montar una máquina virtual Debian 7 y continuar el proceso de instalación en dicha máquina.

6.2.2. Incompatibilidad con la versión de Java

Debian 7 "wheezy"trae por defecto Java versión 6. Durante el proceso de instalación se descubrió que esta versión de Java era incompatible con el software de Ambari. Se decidió instalar Java versión 7 de Oracle. Más adelante durante la instalación tanto en *carissimi* como en *frescobaldi* se encontró la misma situación de incompatibilidad y la solución adoptada fue la misma. Para solventar posteriores problemas asociados a las versiones de Java, se instaló la misma versión en todos los nodos.

```
jmgallego@carissimi:~$ java -version
java version "1.7.0_80"
Java(TM) SE Runtime Environment (build 1.7.0_80-b15)
Java HotSpot(TM) 64-Bit Server VM (build 24.80-b11, mixed mode)
```

FIGURA 6.1: Versión de Java que finalmente se instaló en el servidor carissimi.

6.2.3. Problema con el DNS

Al realizar la instalación del cluster apareció un nuevo problema relacionado con el DNS. La máquina *carissimi* no se encontraba dentro del servidor DNS de la universidad, por tanto no era capaz de resolver las peticiones asociadas a ese nombre de dominio. Este problema afectaba a la comunicación de los dos nodos del cluster, ya que el modo de registro de las máquinas dentro de un cluster se realiza a partir del FQDN. La solución adoptada fue introducir manualmente en ambos nodos la correspondencia entre nombre de dominio de las máquinas y dirección IPv4 dentro del archivo */etc/hosts*.

6.2.4. Permisos del HDFS

Durante la realización de las pruebas, el usuario con el que el *spark-submit* accedía al sistema de ficheros distribuido era el usuario de la sesión,

```
163.117.141.114 carissimi.gast.it.uc3m.es       carissimi
163.117.74.81   debian.jesus                    jesus
163.117.144.158 doc027.lab.it.uc3m.es           doc027
```

FIGURA 6.2: Contenido del archivo /etc/hosts especificando la IP del master Carissimi.

en este caso jmgallego. A la hora de crear el cluster no se especificó ningún usuario con permisos de superusuario por lo que solo se tenía permiso de lectura sobre los archivos. A la hora de lanzar las tareas, se rechazaba la conexión al puerto 8020, que es el puerto utilizado por el namenode. Se adoptaron dos soluciones, siendo la segunda la que finalmente se implementó.

- Para evitar que el programa accediera al hdfs con el usuario jmgallego, que carecía de permisos root en cuanto al hdfs, se forzaba a utilizar el usuario *hdfs*. De esta forma se conseguían los permisos necesarios para poder guardar los resultados de las pruebas. El comando que se añadió al utilizado para lanzar las tareas fue el siguiente:
  - *sudo -u hdfs*
- Como segunda opción, se creó una carpeta en la ruta del hdfs */user/jmgallego* que pertenecía al nuevo usuario *jmgallego*. Se utilizaron los siguientes comandos:
  - *sudo -u hdfs hdfs dfs -mkdir /user/jmgallego*
  - *sudo -u hdfs hdfs dfs -chown jmgallego /user/jmgallego*
  - *sudo -u hdfs hdfs dfs -chgrp jmgallego /user/jmgallego*

6.2.5. Incompatibilidad con máquinas de 32 bits

Al realizar la instalación se observó que el software de Ambari es exclusivo para máquinas con kernels de 64 bits, y no ofrece ninguna solución para instalarlo en máquinas de 32 bits.

En un primer lugar no resultó un problema determinante, simplemente se instaló tanto en la máquina virtual como en *carissimi* y *frescobaldi* un sistema operativo de 64 bits ya que los microprocesadores de esas máquinas eran de 64 bits y se disponía de libertad para cambiar el sistema operativo según los requerimientos y necesidades.

El verdadero problema apareció al intentar instalar dicho software en el laboratorio planteado en el prototipo 2. Las máquinas del laboratorio tenían un sistema operativo de 32 bits, por lo que en un primer momento no era posible la instalación en el cluster del laboratorio.

Con la ayuda de los técnicos del laboratorio, se consiguió instalar el software en las máquinas de los laboratorios a pesar de que la compatibilidad no fue completa, por lo que el despliegue del cluster en el prototipo 2 no fue posible.

Capítulo 7

Planificación, Presupuesto y Marco Legal

7.1. Planificación

La planificación es una parte vital para el éxito de un proyecto. Con la finalidad de explicar la planificación seguida durante la realización del trabajo, primero se van a detallar en un cuadro las tareas que han formado el proyecto acompañadas con su fecha de inicio, de finalización y su duración en días. Para facilitar la comprensión del proceso, se va a acompañar esta información con un diagrama de Gantt que indicará de manera gráfica la duración de cada tarea y la secuencia seguida, así como el tiempo que ha sido dedicado a cada tarea en comparación con el tiempo total del proyecto.

7.1.1. Tabla de planificación

En la siguiente tabla se desglosan las tareas en las que se ha dividido el proyecto. A la hora de realizar el proyecto, se pueden diferenciar tareas de búsqueda de información y redacción de memoria y tareas de instalación y realización de pruebas. La duración total del proyecto ha sido de 31 semanas.

	Nombre de tarea	Duración	Comienzo	Fin
1	Recopilación Bibliografía	134 días	lun 01/02/16	jue 04/08/16
2	Preparación Entorno Virtual Box	11 días	lun 01/02/16	lun 15/02/16
3	Instalación Servidor Virtual Box	11 días	mar 16/02/16	mar 01/03/16
4	Creación Cluster Virtual Box	21 días	lun 07/03/16	lun 04/04/16
5	Instalación Servidor Carissimi	5 días	lun 11/04/16	vie 15/04/16
6	Creación Cluster Carissimi	5 días	lun 18/04/16	vie 22/04/16
7	Instalación Agente Frescobaldi	8 días	lun 18/04/16	mié 27/04/16
8	Planificación Pruebas	2 días	lun 02/05/16	mar 03/05/16
9	Preparación Pruebas	11 días	lun 02/05/16	lun 16/05/16
10	Realización Pruebas Local	15 días	lun 16/05/16	vie 03/06/16
11	Realización Pruebas Cluster	46 días	lun 06/06/16	lun 08/08/16
12	Redacción Memoria	66 días	lun 06/06/16	lun 05/09/16
13	Revisión Memoria	16 días	lun 15/08/16	lun 05/09/16

FIGURA 7.1: Lista de tareas con su respectivas fechas en las que se ha dividido el proyecto.

7.1.2. Diagrama de Gantt

El siguiente diagrama muestra de manera gráfica la distribución temporal de las tareas a lo largo del tiempo total del proyecto.

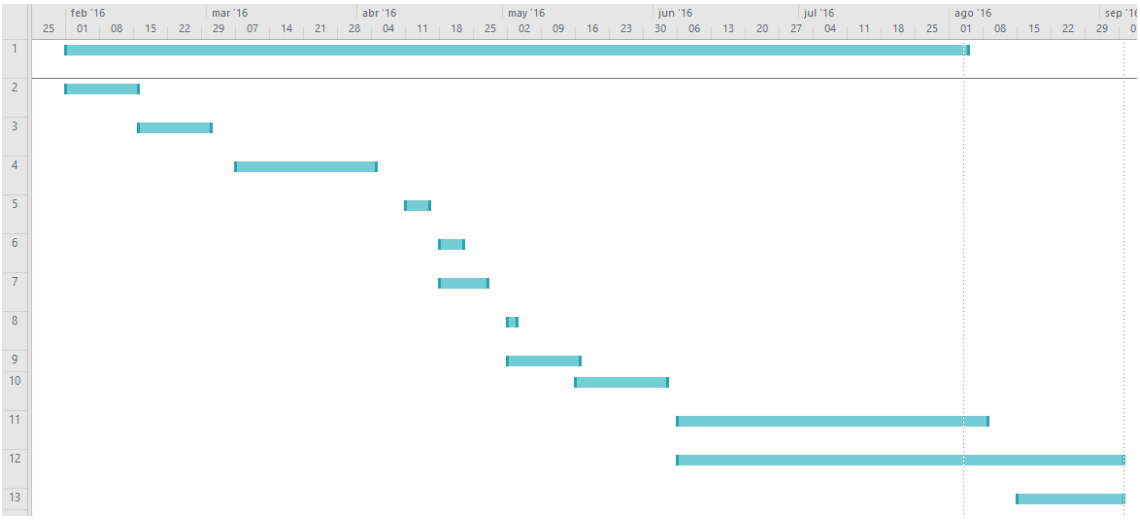


FIGURA 7.2: Tareas del proyecto representada en un diagrama de Gantt.

7.2. Presupuesto

Para realizar una estimación del presupuesto del proyecto, se listará en primer lugar los recursos físicos utilizados en la realización del mismo.

- ***Nodo Master Carissimi:*** Cube Intel 2 Quad Core.
- ***Nodo Esclavo Frescobaldi:*** Servidor Dell Optiplex 740 series n.
- ***Nodo Temporal Jesús:*** Portátil Samsung NP300E5C-T02ES.

Por otra parte, para la realización del proyecto, estos recursos físicos han sido complementados por ciertos softwares. Los Sistemas Operativos y programas utilizados son los siguientes:

- ***Windows 10:*** Utilizado para simular un entorno local de una sola máquina haciendo uso de Virtual Box.
- ***Debian 7:*** Sistema Operativo sobre el que se realiza la instalación del cluster. Utilizado en *Carissimi*.
- ***Ubuntu 14.04:*** Sistema Operativo que ejecuta el nodo esclavo *Frescobaldi*.
- ***Ambari 2.2.0:*** Software utilizado para la creación y monitorización del cluster.
- ***Virtual Box:*** Software de simulación virtual utilizado para realizar un primer acercamiento al software de Ambari.

A estos recursos físicos se le añaden los recursos humanos requeridos para este proyecto. El proyecto ha sido realizado por dos personas con diferentes roles. En primer lugar se ha utilizado un desarrollador para la realización del trabajo y el segundo trabajador ha desempeñado el papel de supervisor del proyecto. El proyecto ha tenido una duración de 31 semanas. La dedicación por parte del desarrollador ha sido de 20 horas semanales. Por su parte, el supervisor ha dedicado 2 horas semanales a las diversas tareas de control y revisión del proyecto.

Si se le asigna un coste de 20€/hora al desarrollador del proyecto y 25 €/hora al supervisor, se puede realizar el coste de recursos humanos del proyecto, mostrado en la tabla 7.2.

El presupuesto asociado a los recursos físicos, tanto hardware como software se desglosa en la tabla 7.1

Recurso	Coste	Horas de vida	Horas de uso	Coste Total
Samsung np300e5c	579 €	26.280 horas	620 horas	13,69 €
Cube Intel 2 Quad Core	398 €	87.600 horas	420 horas	1,91 €
Dell Optiplex 740 series n	298 €	87.600 horas	400 horas	1,36 €
Windows 10	95 €	26.280 horas	620 horas	2,24 €
TOTAL				**19,2 €**

CUADRO 7.1: Presupuesto recursos hardware y software.

Recurso	Horas Dedicadas	Coste por hora	Coste Total
Desarrollador	620 horas	20 €/Hora	12.400 €
Supervisor	62 horas	25 €/Hora	1.550 €
TOTAL			**13.950 €**

CUADRO 7.2: Presupuesto Recursos Humanos.

Recurso	Coste
Recursos Humanos	13.950 €
Recursos Hardware y Sotware	19,2 €
Impuestos Indirectos (21 %)	2.933,53 €
TOTAL	**16.902,73 €**

CUADRO 7.3: Presupuesto final.

El coste total del proyecto una vez añadidos los impuestos indirectos (IVA) aplicando una base imponible del 21 % es de **dieciséis mil novecientos dos euros y setenta y tres céntimos.**

7.3. Marco Legal

No se ha encontrado ninguna legislación aplicable exclusivamente a los laboratorios de Big Data, y por tanto la única ley que se conoce que debe ser cumplida es la ley que afecte a los conjuntos de datos. A nivel de España, debe cumplirse la Ley Orgánica 15/1999 de 13 de diciembre de Protección

de Datos de Carácter Personal (LOPD), que puede considerarse como una de las más restrictivas a nivel mundial.

A nivel europeo se están tomando ciertas medidas en cuanto a la seguridad y la protección legal no en forma de leyes, pero sí en forma de directivas, como la Directiva 96/46/CE de la protección de las personas físicas en el tratamiento de datos personales y su libre circulación y la Directiva 2009/136/CE [28].

Capítulo 8

Conclusions and future work

In this chapter the conclusions of the project will be explained, telling which objectives have been fulfilled and which ones have not. With these conclusions, several ways of future work are presented in case the project is continued.

8.1. Conclusions

It cannot be denied that during the last years Big Data and its possibilities have changed the way of living. With its presence is most of the society aspects, it is very likely to think that the importance of Big Data is each day bigger. So, the knowledge about these technologies has to catch up with the rhythm they advance. This was the principal motivation of the project: to analyze the state of the Big Data tools and understand their behaviour for further application.

The main objective of the project was to implement a cluster with the previously described tools and on the other hand, to write down this process to offer a resource that could be followed and be a base for anyone who want to create a Big Data analysis cluster.

The implementation of the cluster was reached by dividing the task in two parts. In first place, the Ambari software was installed in a virtualized environment and once it was completed, the installation in a cluster with two machines was done, creating a master slave architecture. The principal services for Big Data analysis were added: HDFS, MapReduce, YARN, Spark and ZooKeeper. Those services gave the cluster total functionality at the time of processing any specific task for the computing engines installed, MapReduce and Spark.

The main objective was completed and the correct behaviour of the cluster was checked through the SparkPi test; once that test was completed, it can be confirmed that the first objective was fulfilled.

After this goal, through different tests it was tried to complete the second objective. It was to test the performance of the cluster and check the response of the tools installed depending on the executing mode: local or distributed.

For this measurement, two tests were done in each scenario:

- ***WordCount 500MB*** locally in *carissimi* and distributed in *carissimi* and *frescobaldi*.

- ***WordCount 5,85GB*** locally in *carissimi* and distributed in *carissimi* and *frescobaldi*.

At first glance the result that could be expected from the theory is that the performance of the distributed environment would be greater for those applications more complex. In the study case, the most demanding task is the WordCount of 5,85 GB. Applying the same reasoning, the task that is far from Big Data analysis should have a better performance by executing in local.

The obtained conclusions from these two tests can be summarized in the table C.1:

- ***WordCount 500MB***: In terms of executing time, this task is 5 times faster in local than distributed. Besides, the network usage in lower also in the local case. On the contrary, the work load of the CPU rises in the local tasks compared to the distributed one. This result confirms the idea had from theory. For the task that is not very demanding not only the potential of the Big Data tools is not shown but it is more convenient to execute them locally.
- ***WordCount 5,85GB***: In this case the local task takes more time to complete than the distributed task. In terms of network traffic differences are not visible and it is caused because of the distribution of the task and not the data. The important difference appears in the CPU use. If the task is made in a distributed way the CPU usage is not increased in a significant way, the local execution requires much more CPU. So, it has been proved that despite they have the same execution time, one of the two task is much more heavy for the CPU than the other, being this imperceptible if these metrics are unknown.

	WordCount 500 MB		**WordCount 5,85 GB**	
Execution way	Local	Distributed	Local	Distributed
Time	2 min 16 sec	11 min 42 sec	25 min 35 sec	22 min 55 sec
Network Usage	Lower	Higher	Lower	Higher
CPU Usage	Higher	Lower	Higher	Lower

CUADRO 8.1: Summary of the conclusions after the tests.

The software implementation in a real environment for creating a Big Data laboratory could not be completed. The problems found described in the project's history, which were the incompatibility with 32 bits machines and with the operative system, and a further problem with the permission of certain files, were a obstacle that could not be solved at time. So, even though the installation is almost completed, the third objective could not be fulfilled.

8.2. Future work

In first place, the main further work should be the implementation started in the telematics engineering laboratory, with the goal of creating the

data center described in the design of the project: an external storage system to store the big files, switches to divide the network into several segments and the machines in charge of the computing.

Once completed the installation in the laboratory, this project could be integrated in a class, where by using the authentication supported by Ambari with LDAP, a teacher could create a cluster for a lesson and give access to the student, who could test their application and get feedback about what they are doing. This would complete the educative motivation of this project serving the students who would like to try HDFS, Spark, YARN or any other tool.

Capítulo 9

English Summary

In this chapter, a summary of the whole project will be presented. It will be structured in the same way as the project including the introduction, design, prototype, testing, project's history and conclusions.

9.1. Chapter1: Introduction and Objectives

9.1.1. Introduction

During the last years there has been a change in the way technology is used. It has occupied a place in daily life, and this has generated an unimaginable amount of data. The totality of this data has become a challenge not only for the storage of all the information but also for the extraction of useful conclusions. This challenge requires new techniques and tools for the analysis and also new professionals who could be able to use them.

In order to solve this problem, Big Data analysis comes up and the idea behind this project is to provide a scenario where these tools and techniques can be used and controlled. To work with this technology sets of machines need to be used, called clusters. As it is difficult to have a cluster available, this project is also valid for those who want to test Big Data without such a huge infrastructure.

The relevance that technology has nowadays is proved by taking a look at different data from the Statistics National Institute. According to it, the access to the internet from mobile phones overtakes the access to the internet from computers, and also half of these people also generate data. In market, this relevance is shown in the called High Frequency trading, where Big Data analysis is used to make decisions.

9.1.2. Goals and motivation

The main objective of the project is to implement a cluster capable of executing the tools needed for Big Data analysis. In order to get a cluster with this capability, specific tools must be deployed on it, tools designed for Big Data. Besides, the cluster needs to have the capability of measuring the tasks that are being executed, in order to give a clue to the developers about their applications. To achieve this goal, there will be several metrics that tell information about the state of the cluster. This is set as a second objective, to give the opportunity to developers to know the behaviour of their applications in a controlled scenario.

Moreover, a third objective is to implement the cluster in a real scenario. This implementation will be done at the laboratory of telematics engineering at university Carlos III. Having this set of machines that could compare to a real Big Data laboratory, the performance of a Big Data laboratory could be measured.

9.2. Chapter 2: State of Art

9.2.1. Big Data

The term Big Data can be defined as the set of information that cannot be processed by using traditional techniques and procedures.

The main characteristics that define Big Data are known as V^3: Volume, Variety and Velocity. Volume refers to the amount of data generated. In terms of variety, data can be structured or unstructured, and this shows the variety of the data. As data can be generated and processed in real time, velocity is also a principal characteristic.

To manage all this data there are several free source frameworks that can be chosen: Apache Hadoop, Apache Spark, Apache Flink, Apache Storm and Apache Samza. The principal difference between them is that Hadoop and Spark are aimed to batch processing meanwhile Flink, Storm and Samza are designed to work with real time applications.

9.2.2. Frameworks of Big Data

Inside the framework of Apache Hadoop, there are several modules that can be implemented together in such a way that an ecosystem is created. Even though there exist many modules, in this project only will be studied HDFS, MapReduce, YARN, ZooKeeper and Ambari. There are also other tools such as Pig, Hive HBase, and more.

MapReduce is a computing engine that uses the paradigm key-value to perform operations. By working with enormous sets of these pairs it can be very fast. The main disadvantage is the lack of flexibility at the time of programming the solutions, as they need to adjust to the idea key-value.

HDFS is the distributed file system that is in charge of the storage of all the data. This storage is done across the several nodes of the cluster, that is why it is called distributed.

YARN is the resource manager and scheduler for the tasks that are being executed. It can work on top of HDFS with MapReduce or with other processing module, such as Spark.

ZooKeeper is the module in charge of the distribution of the configuration files. It performs automatically a job that would take a lot of effort to do manually.

Apache Ambari is a tool for monitoring and controlling Hadoop Clusters. It provides metrics for the other components running in the cluster, like the ones listed above.

9.3. Chapter 3: Design

The first thing to be done is to define the design of the cluster that will be implemented. Based on real data centers, the cluster will have external storage systems, switches and servers.

The servers would be in charge of the computing capability of the cluster. As it will be a general purpose cluster, the instances used will also be designed for general purpose.

The switches used are the ones used in the telematics laboratories. These switches from Cisco, named Cisco Small Business SG200-18, give a good switching performance, high fidelity and they are also easy to configure.

An external storage system is needed when the servers storage is not enough. This storage can be used as normal storage for the big files in order just to have the data. At the moment of processing the data, it would be uploaded to the distributed file system installed for example across the nodes of the cluster. By following this schema the storage problem would be solved without having the processing nodes to storage all the data.

The main components of the cluster are the distributed file system and the processing engine, HDFS and MapReduce respectively. To understand the way it works data has to be separated from tasks, as they are treated in different ways.

The data is stored in the DataNodes, being each node of the cluster one. The control of all these is done by the NameNode. There is only one NameNode in each cluster and it distributes and manages all the data in the DataNodes. The NameNode keeps all the information about how data is distributed and which node stores which block of data.

In the same way, the tasks are executed in each node by the TaskTrackers. The one in charge of distributing the tasks is the JobTracker, having in the cluster only one JobTracker and as much TaskTrackers as nodes.

9.4. Chapter 4: Prototype

The project has been divided into two prototypes: the first one is formed by the implementation of a cluster in two servers and the second one is a real implementation in a laboratory of the department of telematics engineering.

9.4.1. First prototype

The first prototype is formed by these two servers:

- ***carissimi.gast.it.uc3m.es***
- ***frescobaldi.gast.it.uc3m.es***

Carissimi is going to be the server, but it also is going to work as an agent. On the other hand, *frescobaldi* is going to act like an agent; so the cluster is going to have one master and two agents.

There are several components installed only in *carissimi* that cover the aspect of the master, which are:

- ***MapReduce History Server:*** Offers an API to get information about the tasks executed by MapReduce.
- ***Ambari Metrics Collector:*** It is a daemon in charge of receiving the metrics from the nodes about several processes.
- ***HDFS NameNode:*** As it has been said, it is the central part of the HDFS and it distributes and controls the data across the nodes of the cluster. Each task has to communicate with the NameNode in order to know in which node is located the data required.
- ***YARN Resource Manager:*** It controls all the resources in the cluster. It works with the Node Manager in each node and with the Application Master of each application.
- ***Spark History Server:*** It is used when Spark works right on top of HDFS without using YARN. In this case, is this component the one in charge of monitoring and scheduling the tasks and the resources of the cluster.

The following components are installed in both servers, and are related with the agent operations.

- ***YARN Node Manager:*** It performs the monitoring tasks of YARN in each node.
- ***HDFS DataNode:*** In charge of storing the data and report periodically to the Name Node the list of blocks they have.
- ***ZooKeeper Server:*** Component that handles the coordination between distributed systems.

In the cluster there are also the Ambari Enhanced Configurations in order to configure the cluster by using a graphical user interface and creating versions of the configurations.

9.4.2. Second Prototype

The second prototype is designed to be implemented in the laboratory 4.1.B01 by using the following machines:

- ***Doc026.lab.it.uc3m.es***
- ***Doc027.lab.it.uc3m.es***
- ***Doc028.lab.it.uc3m.es***
- ***Doc029.lab.it.uc3m.es***

- ***Doc030.lab.it.uc3m.es***

With this distribution, all of the servers would be agents and Doc027 would also be the master. This planning gives 5 agents and 1 master.

During the implementation of the cluster and the installation of the services there have been many problems, these problems will be explained in the history of the project, that a have made impossible the final implementation of the second prototype. Nevertheless, the process is almost complete and it will be successfully done in a short period of time.

9.5. Testing

In order to test the performance of the cluster, Ambari metrics service is going to be used. These metrics give information about CPU usage, network usage, cluster load, memory usage and more.

To check whether everything is working, a test will be executed. It is called SparkPi and it computes an approximation of Pi using the Montecarlo method. The task is completed successfully in 9 minutes and 47 seconds and it tells that everything is working fine.

To test the capability of the cluster there have been scheduled 4 tasks, a WordCount that will be execute in two ways with two different files, this gives the 4 tests. The files are of 500 MB and 5,85 GB and they will be processed in local and cluster mode.

The local test shows a better performance with the smaller file, but it also uses more CPU, meanwhile the distributed task is more effective with large data and it reduces the usage of the CPU.

9.6. Project History

9.6.1. Installation in the laboratory

The implementation in the laboratory has been made working with the technicians of telematics. From the beginning of the installation several problems came out, like the incompatibility with 32 bits kernels and also with the Debian 8 installed in those machines. There were also problems regarding the permissions of certain files, which only had reading access and Ambari needed writing on them.

Finally, due to time constraint, the deployment could not be done totally, but the process is almost finished.

9.6.2. Project problems

As a summary of the problems encountered it can be listed:

- ***Operating System incompatible:*** The software was compatible with Debian 7 or Ubuntu 14.04 but not with Debian 8.

- ***Java version incompatible:*** The Java version of Debian 7 was Java version 6 and it was not compatible so it had to be changed to Java 7.
- ***DNS problem:*** The servers were not in the DNS of the university, so the */etc/hosts* had to be modified.
- ***HDFS permissions:*** A new folder in the HDFS was created to give the user full permissions over the HDFS files.
- ***32 bits kernels incompatible.***

9.7. Conclusions and future work

The principal objective of this project was to implement a cluster equipped with the main tools for Big Data analysis and also to write down that process in order to create a guide for anyone who wants to create his own cluster.

The installation was completed first in a virtual environment and once this installation was done, it was deployed into the scenario with two machines, a master and a slave. The services installed in that cluster to provide the capability of analysis were HDFS, YARN, Spark, ZooKeeper and MapReduce. Those services give the cluster full capability for working with Big Data.

The main objective was fulfilled and it was tested with the SparkPi task. Once that test was completed, it could be confirmed that the main objective of the project was reached.

The rest of the tests were to prove if the second objective had been accomplished. This was to offer a scenario where developers could test their applications and check how they work. The results of the testing was the following in the table 9.1.

	WordCount 500 MB		**WordCount 5,85 GB**	
Execution mode	Local	Distributed	Local	Distributed
Time	2 min 16 sec	11 min 42 sec	25 min 35 sec	22 min 55 sec
Network Usage	Lower	Higher	Lower	Higher
CPU Usage	Higher	Lower	Higher	Lower

CUADRO 9.1: Conclusions summary after the testing..

The 500MB test is even 5 times faster executed locally than in a distributed way and the network usage is also lower. But on the other hand, the CPU usage is higher than in the distributed task. This result shows that this Big Data analysis techniques are not that much efficient with relative small tasks.

For the case of the 6GB WordCount, the execution time is similar but the distributed task is a little faster and the network usage is the same than in the previous test. The main difference is in the CPU usage where the local

task used much more CPU than the distributed one. At this point, can be stated that the distributed process is more efficient.

Having proved this, the second objective was accomplished, it has been implemented a cluster where different metrics of tasks can be measured and compared by developers or students.

As it has been described in the project's history, the third objective has not been accomplished due to the problems that have been found during the process.

9.7.1. Future work

The first line of work is about finishing the implementation in the laboratory. Following the design described in this paper, the prototype set by the prototype 2 and the current work already done in the laboratory 4.1.B01, there is little left for the complete creation of the cluster.

Along with this implementation, the cluster could be used in class using the possibility that Ambari offers of authenticating users by using LDAP. In this way, a teacher could create a cluster a his students could test different apps and understand clearly what is going on in the cluster.

Apéndice A

Código WordCount

En este apéndice se añade el código utilizado para la prueba WordCount. Es un código que ejecuta Spark a través de su cliente ./spark-submit y está programado en Java.

```
package pruebaSpark;

import java.util.Arrays;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFunction;
import scala.Tuple2;

public class prueba {

public static void main(String[] args) {

SparkConf conf = new SparkConf().setAppName("wordCount");
JavaSparkContext sc = new JavaSparkContext(conf);
String input = args[0];
String output = args[1];
JavaRDD<String> textFile = sc.textFile(input);
JavaRDD<String> words =
textFile.flatMap(new FlatMapFunction<String, String>() {
  public Iterable<String> call(String s)
  { return Arrays.asList(s.split(" ")); }
});
JavaPairRDD<String, Integer> pairs =
words.mapToPair(new PairFunction<String, String, Integer>() {
public Tuple2<String, Integer> call(String s)
{ return new Tuple2<String, Integer>(s, 1); }
});
JavaPairRDD<String, Integer> counts =
pairs.reduceByKey(new Function2<Integer, Integer, Integer>() {
public Integer call(Integer a, Integer b) { return a + b; }
});
counts.saveAsTextFile(output);
```

```
}
}
```

El programa admite dos argumentos de entrada que se utilizarán para introducir el path de origen del hdfs al fichero a procesar y el segundo será el path de destino del hdfs del directorio donde se guardarán los resultados, como se observa en la última línea del código. En medio, se realiza el cómputo dividiendo cada palabra mediante un split y se trabaja con los pares clave valor.

Apéndice B

Introducción y objetivos

En este capítulo se introduce al lector el proyecto que se ha realizado, ofreciendo una amplia visión del marco sobre el cual encaja este trabajo. Presenta además los principales objetivos y motivaciones que han hecho que este proyecto se lleve a cabo. Por último, se detalla la estructura para intentar facilitar todo lo posible la lectura de la memoria.

B.1. Introducción

La tecnología, y en concreto los dispositivos TIC, se han vuelto estos últimos tiempos una parte esencial de la sociedad. Para ilustrar este incremento en el uso de la tecnología se presentan los siguientes ejemplos [1]:

- En un mes, en Facebook se comparten treinta mil millones de contenidos, independientemente del tamaño de dichos contenidos.
- Actualmente, toda la música que existe se podría almacenar en un disco duro de unos 500 euros.
- En los dos últimos años se han creado el noventa por ciento de la totalidad de datos almacenados.

Este incremento del uso de la tecnología y las conexiones de red representa a su vez un incremento en la cantidad de datos que alberga la red. El reto para con estos datos no radica únicamente en almacenarlos sino además en la capacidad de extraer información útil de ellos, y en eso se basa el análisis de Big Data.

Para poder almacenar y analizar esta enorme cantidad de datos se necesitan herramientas específicas diseñadas para tal fin, así como profesionales cualificados para poder analizar y así evitar la formación de las conocidas tumbas de datos. Tras esta idea surge este proyecto, intentando acercar las herramientas para el análisis de Big Data a entornos fácilmente desplegables con fines educativos.

Estas herramientas utilizadas para el análisis trabajan sobre grupos de máquinas, aprovechando de manera conjunta los recursos que cada máquina puede ofrecer individualmente. No obstante, disponer de un cluster de máquinas no es por lo general sencillo, y es por ese motivo por el cuál cobra especial importancia el estudio sobre las diferentes maneras en las que un cluster se debe manejar.

Una de las principales intenciones que se busca con este trabajo es poner en una balanza las dificultades que se encuentran para conseguir el control de un cluster y las oportunidades de negocio o docentes que supone disponer de tal infraestructura.

Por último, resulta interesante poder estudiar las herramientas típicas del análisis de Big Data a pesar de no contar con tal infraestructura, esto es, ser capaces de interactuar con ellas en un entorno formado por un solo ordenador y así entender el funcionamiento y las limitaciones que esto conlleva.

B.2. Entorno socioeconómico

De acuerdo con el **Instituto Nacional de Estadística** a fecha de 2 de Octubre de 2014, el 74,4 % de los hogares disponía de conexión a internet. Por primera vez en España, esta conexión no se realizaba exclusivamente desde ordenadores, sino que el acceso a internet a través de dispositivos móviles superó al acceso a través de ordenadores personales, 77,1 % frente a 73,3 %, respectivamente [2].

No sólo se ha incrementado el acceso a internet, sino también ha cambiado la forma en la que se utiliza la conexión. El 51,1 % de la población es activa en redes sociales, esto es, la mitad de la población no sólo consume contenido sino que además lo genera. Estos datos muestran el cambio social que está implicando el cambio de la tecnología, y viceversa.

El Big Data ha revolucionado también el ámbito de la gestión empresarial. Aquellas estrategias y herramientas que usan las empresas para su crecimiento se conocen como Business Intelligence, y está muy relacionado con el Big Data en los últimos tiempos. Por poner un ejemplo concreto, la mayoría de las inversiones que se realizan en bolsa son realizadas por decisiones ejecutadas por algoritmos de estimación y decisión que corren sobre clusters de Big Data, debido a la necesidad de procesamiento en tiempo real. Estas transacciones se conocen como operaciones de alta frecuencia, o High Frecuency trading y representan el 50 % de los volúmenes comerciales de las bolsas de EEUU [3].

B.3. Objetivos y motivación

El principal objetivo que se persigue en este proyecto es la implementación de un cluster dotado de las herramientas más populares para el análisis de Big Data. Se pretende que esta implementación se convierta en un proceso guiado, intuitivo y fácil para todo aquél que, disponiendo de unos recursos mínimos, quiera utilizar y aprender nuevos conceptos de este campo.

Este objetivo es de gran relevancia en el ámbito docente, con el fin de que un grupo de alumnos tengan la posibilidad de experimentar por su cuenta el uso de herramientas utilizadas en el ámbito profesional; siendo monitorizadas y controladas por un tutor.

Además de este objetivo, también se pretende proporcionar un entorno a desarrolladores en el que puedan probar sus aplicaciones con el fin de comprobar su correcto funcionamiento. Para poder medir el funcionamiento de las aplicaciones se debería contar con parámetros de ejecución o sondas capaces de ofrecer métricas que muestren qué está sucediendo realmente en las máquinas. Las posibilidades ofrecidas irían desde la opción de comprobar el efecto de la programación de las tareas, pasando por las diferentes configuraciones que se pueden aplicar a un cluster y sus efectos hasta las diferencias entre la ejecución en un cluster frente a una ejecución local de las mismas tareas.

Adicionalmente, y de manera complementaria al objetivo principal del proyecto que es la implementación del cluster, se quiere realizar esta implementación en un entorno lo más real posible. Este entorno estaría formado por un laboratorio real con varias decenas de máquinas e incluso diferentes segmentos de red. Este objetivo nos daría una visión general del funcionamiento de un cluster formado por un gran número de máquinas y completaría la motivación docente del proyecto. Para completar este objetivo, se plantea la implementación de un cluster en los laboratorios del departamento de ingeniería telemática de la universidad Carlos III de Madrid.

B.4. Contenido de la memoria

Este documento consta de 9 capítulos y tres apéndices. Se va a proceder a resumir el contenido de cada capítulo para facilitar la lectura y comprensión de la memoria.

B.4.1. Capítulo 1: Introducción y objetivos

Este capítulo sirve de introducción y expone la motivación y principales objetivos del proyecto, así como la estructura de la memoria.

B.4.2. Capítulo 2: Estado del arte

En este capítulo se ofrece una visión general del estado del Big Data en la actualidad. Las posibilidades que nos brinda el análisis de Big Data y algunas herramientas de código libre que se pueden utilizar para exprimir esas posibilidades.

B.4.3. Capítulo 3: Diseño

Este capítulo detalla la idea original que se planteó a la hora de realizar el proyecto. Plantea el diseño de un cluster con los componentes necesarios para que sea completamente funcional en un entorno real de trabajo, como puede ser una empresa.

B.4.4. Capítulo 4: Prototipo

Este capítulo cubre la descripción del cluster creado mostrando los detalles tanto de los componentes instalados como del proceso de instalación.

B.4.5. Capítulo 5: Pruebas

Este capítulo se pone a prueba el cluster creado mediante diferentes tests y detalla la respuesta del cluster ante diferentes tareas.

B.4.6. Capítulo 6: Historia del proyecto

En este capítulo se explican los diferentes problemas que han aparecido durante la realización del proyecto, tanto en la instalación como en la parte final de las pruebas.

B.4.7. Capítulo 7: Planificación, presupuesto y marco legal

En este capítulo se detalla la organización que se ha seguido durante el trabajo así como el listado de los recursos utilizados. A esta lista de recursos se le ha añadido el presupuesto del proyecto.

B.4.8. Capítulo 8: Conclusiones y líneas futuras

Este capítulo compara los objetivos iniciales con los resultados finales extrayendo unas conclusiones del trabajo realizado. Además incluye unas guías y recomendaciones de hacia dónde puede evolucionar el proyecto si se diera la opción de continuarlo.

B.4.9. Capítulo 9: Resumen en inglés

Este capítulo es un resumen de las principales partes del proyecto, que son la introducción, el estado del arte, el diseño, el prototipo, la historia del proyecto y las conclusiones.

B.4.10. Apéndice A: Código del WordCount

En este apéndice se añade el código usado para probar el cluster.

B.4.11. Apéndice B: Introducción

En este apéndice se presenta la introducción del proyecto en español.

B.4.12. Apéndice C: Conclusiones y líneas futuras

En este apéndice se presentan tanto las conclusiones como las lineas futuras en español.

Apéndice C

Conclusiones y líneas futuras

En este capítulo se van a redactar las conclusiones del proyecto, explicándose qué objetivos iniciales se han cumplido y cuáles no ha sido posible cumplir. Con estas conclusiones se presenta el final del proyecto acompañado de unas líneas futuras que indican el camino que podría seguir una continuación del mismo.

C.1. Conclusiones

Es un hecho innegable que durante los últimos años el Big Data y sus posibilidades han cambiado la forma en la que se vive. Con su presencia actual de la en la mayoría de aspectos de la sociedad, no resulta descabellado aceptar que la importancia que el Big Data tiene es cada vez mayor. Por tanto, el conocimiento general sobre estas tecnologías debe acompañar de manera natural al ritmo al que estas avanzan. Esta era la principal motivación del estudio: analizar el estado actual de las tecnologías y herramientas de Big Data y comprender su funcionamiento para su posterior aplicación.

El principal objetivo del proyecto era por un lado implementar un cluster con dichas herramientas y por otro lado, documentar ese proceso para ofrecer un recurso que pueda ser seguido y servir de guía a quién quiera contar con su propio grupo de máquinas dotado de capacidad de análisis de Big Data.

La implementación del cluster se consiguió dividiendo la tarea en dos partes. En primer lugar se instaló el software de Ambari en un entorno virtualizado y una vez se hubo completado con éxito se procedió a la instalación en un entorno real con dos máquinas formando un escenario de esclavo y master. El cluster se consiguió crear añadiendo los servicios fundamentales para el análisis de Big Data: HDFS, MapReduce, YARN, Spark y ZooKeeper. Dichos servicios dotaban al cluster de total funcionalidad a la hora de procesar cualquier tarea específica para alguno de los dos motores de cómputo instalados, que son MapReduce y Spark.

El objetivo principal se consiguió y se comprobó el correcto funcionamiento del cluster a través de la prueba SparkPi; una vez esa prueba se ha completado, se puede afirmar que se ha alcanzado el primer objetivo.

Tras este primer objetivo, mediante la realización de las diversas pruebas se intentó completar el segundo objetivo. Este consistía en poner a prueba el rendimiento del cluster a través de diferentes tareas y comprobar la

respuesta de las herramientas instaladas a través de un cluster o a través de una sola máquina.

Para esta medición se realizaron dos pruebas en cada escenario:

- ***WordCount 500MB*** local en *carissimi* y distribuido en *carissimi* y *frescobaldi*.
- ***WordCount 5,85GB*** local en *carissimi* y distribuido en *carissimi* y *frescobaldi*.

A priori, el resultado que cabría esperar a partir de la teoría sería el mayor rendimiento del entorno distribuido para aquellas tareas más exigentes. En el caso de estudio, la tarea más exigente es aquella de 5,85 GB. Aplicando el mismo razonamiento, la tarea que más se aleja de lo que sería un ejercicio de Big Data debería tener un mayor rendimiento al realizarse de manera local.

Las conclusiones obtenidas de estas dos pruebas se pueden resumir de la siguiente forma, además de lo que se muestra en la tabla C.1 :

- ***WordCount 500MB***: En cuanto a tiempo de ejecución, esta tarea es hasta 5 veces más rápida de manera local que de manera distribuida. Además, el uso de la red es menor también en el escenario local. Por el contrario, la carga de trabajo de la CPU aumenta en la tarea local frente a la tarea distribuida. Este resultado confirma la intuición que se tenía previamente. Para aquella tarea que sea poco exigente no solo no se aprecia el potencial de las herramientas de Big Data sino que es más conveniente realizar la tarea de forma local.
- ***WordCount 5,85GB***: Al contrario que en el caso anterior, la tarea local tarda más en completarse que la tarea distribuida. En el aspecto de tráfico de red no se aprecian diferencias con la prueba anterior ya que el funcionamiento es el mismo independientemente de la cantidad de texto a procesar, debido a la idea de distribuir la tarea y no los datos. La diferencia sustancial aparece en el uso de la CPU. Mientras que al realizar la tarea de manera distribuida no se aprecia una subida significativa del porcentaje de uso del procesador, en la ejecución local se exige mucha más capacidad de cómputo. Por tanto, queda demostrado que a pesar de tener unos tiempos similares de ejecución, una de las dos tareas resulta mucho más ligera para la máquina, siendo esto imperceptible si no se conocen estas métricas.

La implementación del software en un entorno real para crear un laboratorio Big Data no pudo completarse. Los problemas encontrados descritos en la historia del proyecto, que fueron la incompatibilidad del software con máquinas de 32 bits y con la versión del sistema operativo y más adelante un problema definitivo de permisos sobre determinados ficheros, sumados a las limitaciones temporales del proyecto supusieron un obstáculo que no pudo salvarse. Por tanto, a pesar de tener casi terminada la instalación y faltando muy poco para su realización, no se ha podido implementar el cluster en el laboratorio tal y como se tenía pensado.

	WordCount 500 MB		WordCount 5,85 GB	
Modo ejecución	Local	Distribuido	Local	Distribuido
Tiempo	2 min 16 seg	11 min 42 seg	25 min 35 seg	22 min 55 seg
Uso de red	Menor	Mayor	Menor	Mayor
Uso CPU	Mayor	Menor	Mayor	Menor

CUADRO C.1: Resumen de las conclusiones después de las pruebas..

C.2. Líneas futuras

En primer lugar, la principal línea de investigación que debería seguir el proyecto es la implementación empezada en el laboratorio con el objetivo de conseguir crear un centro de datos con la misma estructura que la explicada en el diseño del proyecto. Esta constaría de las máquinas propias del laboratorio, un almacenamiento externo para guardar los archivos de Big Data y diferentes segmentos de red, o una virtualización de esos segmentos.

Una vez conseguido el laboratorio, se podría integrar dentro de un entorno de clase, donde mediante el uso de los servidores de autenticación LDAP el profesor pueda dar acceso a los estudiantes a un cluster que él haya creado previamente. De esta forma, se completaría la finalidad docente de este proyecto, al servir en su totalidad a los alumnos que quieran experimentar con Spark, HDFS, YARN o demás herramientas.

Bibliografía

[1] A. Monleón-Getino, *El impacto del big-data en la sociedad de la información. significado y utilidad*, 2015.

[2] I. N. de Estadística, *Encuesta sobre equipamiento y uso de tecnologías de información y comunicación en los hogares. año 2014.* 2014. dirección: http://www.ine.es/prensa/np864.pdf.

[3] E. economista, *Big data: ¿el nuevo santo grial de la inversión?*, 2016. dirección: https://goo.gl/He0GSE.

[4] R. B. Fragoso. (2012). ¿qué es big data?, dirección: www.ibm.com/developerworks/ssa/local/im/que-es-big-data (visitado 10-06-2016).

[5] D. d. Paul C. Zikopoulos C. Eaton, *Understanding big data: Analytics for enterprise class hadoop and straming data*. McGraw-Hill, 2012.

[6] M. Mayo. (2016). Top big data processing frameworks, dirección: www.kdnuggets.com/2016/03/top-big-data-processing-frameworks.html.

[7] T. A. S. Foundation. (2014). Welcome to apache hadoop, dirección: http://hadoop.apache.org/ (visitado 14-06-2016).

[8] Databricks. (2016). Apache spark ecosystem, dirección: https://databricks.com/spark/about (visitado 14-06-2016).

[9] T. A. S. Foundation. (2015). Apache flink, dirección: https://flink.apache.org/ (visitado 14-06-2016).

[10] ——, (2015). Apache storm, dirección: http://storm.apache.org/ (visitado 14-06-2016).

[11] ——, (2015). ¿what is samza?, dirección: http://samza.apache.org/ (visitado 14-06-2016).

[12] S. Seshachala. (2015). Bigdata – understanding hadoop and its ecosystem, dirección: http://devops.com/2015/06/01/bigdata-understanding-hadoop-ecosystem/ (visitado 14-06-2016).

[13] Hortonworks. (2016). Apache hadoop mapreduce, dirección: http://hortonworks.com/apache/mapreduce/ (visitado 16-06-2016).

[14] ——, (2016). Apache hadoop yarn, dirección: http://hortonworks.com/apache/yarn/ (visitado 19-06-2016).

[15] T. White, *Hadoop: The definitive guide*, 2010.

[16] Hortonworks. (2016). Apache hive, dirección: http://hortonworks.com/apache/hive/ (visitado 16-06-2016).

[17] ——, (2016). Apache hbase, dirección: http://hortonworks.com/apache/hbase/ (visitado 16-06-2016).

[18] J. Hanna. (2016). Zookeeper overview, dirección: https://wiki.apache.org/hadoop/ZooKeeper/ProjectDescription (visitado 16-06-2016).

[19] Hortonworks. (2016). Apache ambari, dirección: https://hortonworks.com/apache/ambari/ (visitado 19-06-2016).

[20] Cisco. (2016). Cisco 200 series switches data sheet, dirección: http://goo.gl/ZUe6FP (visitado 30-06-2016).

[21] D. I. Telemática. (2016). Cableado laboratorios departamento ingeniería telemática, dirección: http://goo.gl/z2qXjD (visitado 30-06-2016).

[22] S. Wagle. (2016). Metrics collector api specification, dirección: http://goo.gl/vZexVT (visitado 06-07-2016).

[23] S. R. k. Shvachko H. Kuang y R. Chansler, *The hadoop distributed file system*, 2010.

[24] V. K. Vavilapalli. (2012). Apache hadoop yarn resourcemanager, dirección: http://hortonworks.com/blog/apache-hadoop-yarn-resourcemanager/ (visitado 20-07-2016).

[25] ——, (2012). Apache hadoop yarn nodemanager, dirección: http://hortonworks.com/blog/apache-hadoop-yarn-nodemanager/ (visitado 22-07-2016).

[26] F. Junqueira y B. Reed, *Zookeeper: Distributed process coordination*, 2013.

[27] S. Gunturi. (2016). Enhanced configs, dirección: https://cwiki.apache.org/confluence/display/AMBARI/Enhanced+Configs (visitado 26-07-2016).

[28] M. Ángel Pantoja, *Implicaciones de seguridad de big data*, 2015. dirección: http://goo.gl/f81IvF.